

深度学习与R语言

程显毅　施　佺　编著



提供源程序代码

下载网址为 www.cmpbook.com



机械工业出版社
CHINA MACHINE PRESS

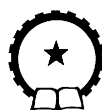


VISIT…

LANZAROTE Caliente.COM

深度学习与 R 语言

程显毅　施　佺　编著



机 械 工 业 出 版 社

近年来，深度学习可谓是机器学习方向的明星概念，不同的深度学习模型分别在图像处理与自然语言处理等任务中取得了前所未有的好成绩。

在许多场合都有这样的需求“如何对感兴趣的领域快速理解和使用深度学习技术?”答案涉及复杂的数学、编程语言（如 C、C++和 Java）。但随着 R 的兴起，现在使用深度学习技术比以往更容易。因为 R 易学易用，不要求很扎实的编程基础，它被广泛地应用于机器学习实践和教学中。即使对 R 语言不是很了解的用户也可以通过一些包来搭建深度学习网络。

全书 11 章，分为原理篇（第 1~8 章）和应用篇（第 9~11 章）。原理篇按照深度学习的发展过程，主要讨论了浅层神经网络、深度神经网络、卷积神经网络、递归神经网络、自编码网络、受限玻耳兹曼机和深度置信网。应用篇讨论 R 环境部署深度学习环境的一些策略，包括：MXNetR、H2O 和其他深度学习 R 包以及一些典型的应用。

本书可用作本科高年级机器学习课程参考书或数据科学课程教材，也可供对人工智能、机器学习感兴趣的读者参考阅读。

图书在版编目（CIP）数据

深度学习与 R 语言/程显毅，施佺编著.—北京：机械工业出版社，2017.6
ISBN 978-7-111-57073-8

Ⅰ.①深… Ⅱ.①程… ②施… Ⅲ.①学习（人工智能） ②程序语言－程序设计 Ⅳ.①TP18 ②TP312

中国版本图书馆 CIP 数据核字（2017）第 125518 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
策划编辑：汤 枫 责任编辑：汤 枫
责任校对：张艳霞 责任印制：常天培

涿州市京南印刷厂印刷

2017 年 6 月第 1 版・第 1 次印刷
184mm×260mm・13.5 印张・318 千字
0001-4200 册
标准书号：ISBN 978-7-111-57073-8
定价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
服务咨询热线：（010）88361066	机 工 官 网：www.cmpbook.com
读者购书热线：（010）68326294	机 工 官 博：weibo.com/cmp1952
（010）88379203	教育服务网：www.cmpedu.com
封面无防伪标均为盗版	金 书 网：www.golden-book.com

前　　言

深度学习是机器学习领域一个新的研究方向，近年来在语音识别、计算机视觉等多类应用中取得突破性的进展，其动机在于建立模型模拟人类大脑的神经连接结构，进而给出数据的解释。

深度学习之所以被称为“深度”，是相对支持向量机（Support Vector Machine，SVM）、提升方法（Boosting）、最大熵方法等“浅层学习”方法而言的。浅层学习依靠人工经验抽取样本特征，网络模型学习后获得的是没有层次结构的单层特征；而深度学习通过对原始信号进行逐层特征变换，将样本在原空间的特征表示变换到新的特征空间，自动地学习得到层次化的特征表示，从而更有利于分类或特征的可视化。

本书的目的是把强大的深度学习技术传递到想实践深度学习的读者手中，而不是让读者理解深度学习的理论细节。因此，内容重点是数据分析和建模，注意力完全集中在能有效工作的深度学习技术、理念和策略上，这样可以用最少的时间快速消化和部署深度学习应用。

本书具有以下特点：

1）让读者清楚如何在 R 中使用深度学习。书中给出了大量的深度学习应用案例，这些例子可以直接输入到 R 环境中运行，指导读者一步一步构建和部署深度学习模型。

2）深度学习不需要很深的数学基础作为前提。无论你是谁？无论你来自哪里？无论你的受教育背景如何？都有能力使用这本书中论述的方法。

3）每一章都提供了进一步学习的详细参考资料，并且大部分是免费的。

图的上半部分给出了本书的学习路线，第 1、5、7 章相对独立，是学习深度神经网络的基础，虚线表示分类，实线表示支持，核心是第 3、4、8 章，每一章下面的英文表示依赖的 R 包。图的下半部分是深度学习 R 包与各章的关系。

本书主要参考了 N. D. Lewis 所著的《Deep Learning Made Easy with R——A Gentle Introduction for Data Science》，在此表示感谢。感谢研究生谢璐、胡海涛、周春瑜、姚泽峰和沈佳杰在材料整理方面所做的工作。感谢胡彬、陈晓勇、李跃华老师

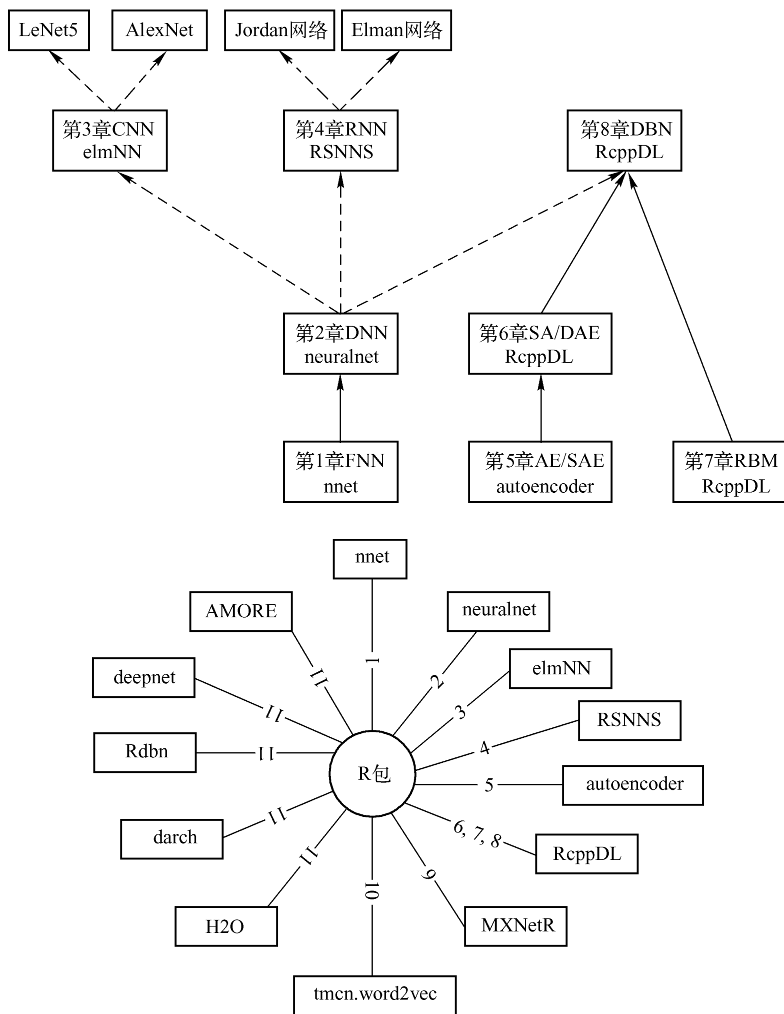


图　深度学习 R 包与各章的关系

对本书提出的宝贵意见。

本书的出版得到南通大学 – 南通智能信息技术联合研究中心开放课题项目的资助和南通大学学术著作出版基金资助。

深度学习领域发展迅猛，对许多问题作者并未做深入研究，一些有价值的新内容也来不及收入本书。加上作者知识水平和实践经验有限，书中难免存在不足之处，敬请读者批评指正。

编　者

目　　录

前言
第 1 章　引言 …… 1
1.1　关于深度学习 …… 1
1.1.1　深度学习兴起的渊源 …… 1
1.1.2　深度学习总体框架 …… 3
1.1.3　深度学习本质 …… 4
1.1.4　深度学习应用 …… 5
1.2　前向反馈神经网络 FNN …… 5
1.2.1　多层感知器 …… 5
1.2.2　神经元的作用 …… 6
1.2.3　激活函数 …… 8
1.2.4　学习算法 …… 9
1.3　R 语言基础 …… 11
1.3.1　入门 …… 11
1.3.2　基本语法 …… 13
1.3.3　数据 …… 14
1.3.4　绘图 …… 19
1.3.5　数据准备 …… 21
1.3.6　基本运算 …… 23
1.4　FNN 的 R 实现 …… 23
1.5　学习指南 …… 27
第 2 章　深度神经网络 DNN …… 28
2.1　DNN 原理 …… 28
2.2　DNN 应用 …… 30
2.2.1　提高雾天视觉能见度 …… 31
2.2.2　打击黑客和网络犯罪 …… 31

2.2.3 图像压缩 …… 33
2.2.4 函数逼近 …… 34
2.3 DNN 应用需要注意的一些问题 …… 38
2.3.1 神经元数量 …… 38
2.3.2 最佳层数的选择 …… 39
2.3.3 训练时间过长 …… 39
2.3.4 过拟合 …… 40
2.4 DNN 应用技巧 …… 40
2.5 单响应变量 DNN 的 R 实现 …… 42
2.6 多响应变量 DNN 的 R 实现 …… 46
2.7 学习指南 …… 51
第 3 章 卷积神经网络 CNN …… 52
3.1 CNN 原理 …… 52
3.1.1 局部感知 …… 53
3.1.2 权值共享 …… 54
3.1.3 多卷积核 …… 54
3.1.4 池化 …… 56
3.2 多层卷积 …… 57
3.2.1 ImageNet - 2010 网络结构 …… 57
3.2.2 DeepID 网络结构 …… 58
3.3 CNN 的 R 实现 …… 58
3.4 学习指南 …… 59
第 4 章 递归神经网络 RNN …… 61
4.1 RNN 原理 …… 61
4.2 Elman 网络 …… 61
4.2.1 承接层神经元的作用 …… 62
4.2.2 信息流动 …… 62
4.2.3 Elman 网络应用 …… 63
4.3 Jordan 网络 …… 65
4.3.1 Jordan 网络结构 …… 65
4.3.2 Jordan 网络应用 …… 65

4.4 RNN 的 R 实现 …… 66
4.5 学习指南 …… 77
第 5 章 自编码网络 AE …… 79
5.1 无监督学习过程 …… 79
5.2 AE 基本结构 …… 80
5.2.1 降维问题 …… 81
5.2.2 特征抽取 …… 82
5.3 稀疏自动编码网络 SAE …… 83
5.3.1 Kullback - Leibler 散度 …… 84
5.3.2 使用 SAE 注意事项 …… 84
5.4 SAE 的 R 实现 …… 85
5.5 学习指南 …… 94
第 6 章 堆栈自编码网络 SA …… 95
6.1 SA 原理 …… 96
6.2 SA 的 R 实现 …… 97
6.3 降噪自编码网络 DAE …… 99
6.3.1 随机掩蔽的椒盐噪声 …… 100
6.3.2 DAE 基本任务 …… 100
6.3.3 标准化堆栈降噪自编码网络 …… 100
6.4 DAE 的 R 实现 …… 101
6.5 学习指南 …… 105
第 7 章 受限玻耳兹曼机 RBM …… 107
7.1 RBM 原理 …… 107
7.1.1 玻耳兹曼机的四类知识 …… 107
7.1.2 能量和概率的作用 …… 108
7.1.3 联合概率分布表示的自编码网络 …… 109
7.1.4 模型学习的目标 …… 109
7.2 训练技巧 …… 110
7.2.1 技巧 1：Gibbs 采样 …… 110
7.2.2 技巧 2：最小化 KL 距离 …… 110
7.2.3 技巧 3：使用 RLU 激活函数 …… 111

7.2.4 技巧 4：模拟退火 …… 111
7.3 对深度学习的质疑 …… 112
7.4 RBM 应用 …… 112
7.4.1 肝癌分类的 RBM …… 113
7.4.2 麻醉镇定作用预测的 RBM …… 114
7.5 RBM 的 R 实现 …… 115
7.6 学习指南 …… 118
第 8 章 深度置信网络 DBN …… 120
8.1 DBN 原理 …… 120
8.2 应用案例 …… 121
8.3 DBN 的 R 实现 …… 123
8.4 学习指南 …… 126
第 9 章 MXNetR …… 128
9.1 MXNet 技术特性 …… 128
9.2 MXNetR 安装 …… 129
9.2.1 安装 MXNet 基本需求 …… 129
9.2.2 MXNet 云设置 …… 130
9.2.3 MXNet 安装方法 …… 130
9.2.4 MXNetR 安装方法 …… 131
9.2.5 常见的安装问题 …… 132
9.3 MXNetR 在深度学习中的应用 …… 133
9.3.1 二分类模型 …… 133
9.3.2 回归模型与自定义神经网络 …… 135
9.3.3 手写数字竞赛 …… 137
9.3.4 图像识别应用 …… 141
9.4 学习指南 …… 143
第 10 章 word2vec 的 R 语言实现 …… 144
10.1 word2vec 词向量由来 …… 144
10.1.1 统计语言模型 …… 144
10.1.2 神经网络概率语言模型 …… 145
10.2 word2vec——词向量特征提取模型 …… 145

10.2.1 词向量 …… 145
10.2.2 CBOW 的分层网络结构——HCBOW …… 146
10.2.3 word2vec 流程 …… 150
10.3 word2vec 的 R 实现 …… 151
10.3.1 tmcn. word2vec 包 …… 151
10.3.2 word2vec 自编译函数 …… 153
10.3.3 使用 tmcn. word2vec 和 word2vec 注意的问题 …… 154
10.4 学习指南 …… 155
第 11 章 R 语言其他深度学习包 …… 156
11.1 darch 包 …… 156
11.2 Rdbn 包 …… 161
11.2.1 Rdbn 原理 …… 161
11.2.2 Rdbn 安装 …… 161
11.2.3 Rdbn 应用 …… 162
11.3 H2O 包 …… 164
11.3.1 H2O 原理 …… 164
11.3.2 H2O 应用 …… 167
11.4 deepnet 包 …… 169
11.5 mbench 包 …… 171
11.6 AMORE 包 …… 175
11.7 学习指南 …… 177
附录 …… 178
附录 A 深度学习发展史 …… 178
附录 B 深度学习的未来——GAN …… 195
附录 C R 包分类 …… 198
参考文献 …… 201
后记 …… 204

第 1 章　引　　言

1.1　关于深度学习

机器学习是人工智能的一个分支，在很多时候几乎成为人工智能的代名词。简单来说，机器学习就是通过算法使得机器能从大量历史数据中学习规律，从而对新的样本做智能识别或对未来做预测。从 20 世纪 80 年代末期以来，机器学习的发展大致经历了两次浪潮：浅层学习(Shallow Learning)和深度学习(Deep Learning)。

1.1.1　深度学习兴起的渊源

深度学习起源于对神经网络的研究，20 世纪 60 年代，受神经科学对人脑结构研究的启发，为了让机器也具有类似人一样的智能，人工神经网络被提出用于模拟人脑处理数据的流程。最著名的学习算法称为感知机。但随后人们发现，两层结构的感知机模型不包含隐层单元，输入是人工预先选择好的特征，输出是预测的分类结果，因此只能用于学习固定特征的线性函数，而无法处理非线性分类问题。Minsky 等指出了感知机的这一局限，由于当时其他人工智能研究学派的抵触等原因，使得对神经网络的研究遭受到巨大的打击，陷入低谷。直到 20 世纪 80 年代中期，反向传播（Back Propagation，BP）算法的提出，提供了一条如何学习含有多隐层结构的神经网络模型的途径，让神经网络研究得以复苏。

由于增加了隐层单元，多层神经网络比感知机具有更灵活且更丰富的表达力，可以用于建立更复杂的数学模型，但同时也增加了模型学习的难度，特别是当包含的隐层数量增加的时候，使用 BP 算法训练网络模型时，常常会陷入局部最小值，而在计算每层结点梯度时，在网络低层方向会出现梯度衰竭的现象。因此，训练含有许多隐层的深度神经网络一直存在困难，导致神经网络模型的深度受到限制，制约了其性能。

2006年之前，大多数机器学习仍然在探索浅层结构（Shallow－structured），这种结构包含了一层典型的非线性特征变换的单层，而缺乏自适应非线性特征的多层结构。如隐马尔可夫模型（HMM）、线性或非线性动态系统、条件随机域（CRFs）、最大熵（Max－entropy）模型、支持向量机（SVM）、逻辑回归、内核回归和具有单层隐藏层的多层感知器（MLP）神经网络。这些浅层学习模型的共性是由仅有的单层组成的简单架构负责转换原始输入信号或输入特征为特定问题特征空间时，其过程不可观察。以支持向量机为例，它是一种浅层线性独立模型，当使用核技巧时，SVM具有一个特征转换层，否则特征转换层个数为0。浅层架构在许多简单或受限问题中，早已被证明卓有成效，但是由于它们的有限建模与表现能力，导致在处理涉及自然信号如人的讲话、自然的声音和语言、自然的图像和视觉场景等更为复杂的现实应用时，产生了困难。

在实际应用中，如对象分类问题（对象可以是文档、图像、音频等），人们不得不面对的一个问题是如何用数据来表示这个对象，当然这里的数据并非初始的像素或者文字，也就是这些数据是比初始数据具有更为高层的含义，这里的数据往往指的是对象的特征。例如人们常常将文档、网页等数据用词的集合来表示，根据文档的词集合表示到一个词组短语的向量空间（Vector Space Model，VSM）中，然后才能根据不同的学习方法设计出适用的分类器来对目标对象进行分类。因此，选取什么特征或者用什么特征来表示某一对象对于解决一个实际问题非常得重要。然而，人为地选取特征的时间代价是非常昂贵的。而所谓的启发式算法得到的结果往往不稳定，结果的好坏经常是依靠经验和运气。于是，人们考虑到利用自动学习来完成特征抽取这一任务。深度学习的产生就是缘于此任务，它又被称为无监督的特征学习，从这个名称就可以知道这是一个没有人为参与的特征选取方法。

深度结构学习，或者通常更多人称之为深度学习，从2006年开始作为一个新兴的领域出现在机器学习的研究中。深度学习的概念是2006年由Geoffrey Hinton等人在《Science》上发表的一篇文章*Reducing the dimen sionality of data with neural networks*提出来的，开启了深度学习在学术界和工业界的浪潮。这篇文章有两个主要观点：①多隐层的人工神经网络具有优异的特征学习能力，学习得到的特征对数据有更本质的刻画，从而有利于可视化或分类；②深度神经网络在训练上的难度，可以通过"逐层初始化"来有效克服，在这篇文章中，逐层初始化是通过无监督学习实现的。2006年的另外3篇论文[2－4]改变了训练深度架构失败的状况，由Hinton革命

性地在深度置信网络（Deep Belief Networks，DBN）上的工作所引领。

1.1.2　深度学习总体框架

随着计算机处理器速度越来越快、存储器容量越来越大以及数据表达的形式多样化，各类大小事务都能使用深度学习进行实时数据分析。

深度学习是利用多层感知机学习模型来对数据进行有监督或者无监督学习。模型中的层是由多段非线性数据变换构成的，层次越高，对数据特征的表达就越抽象。

图 1.1 说明了深度学习的金字塔结构。在底部是数据学习的两种核心形式：有监督学习和无监督学习。核心要素非线性变换位于金字塔结构的中心，金字塔上部是各种各样的神经网络。

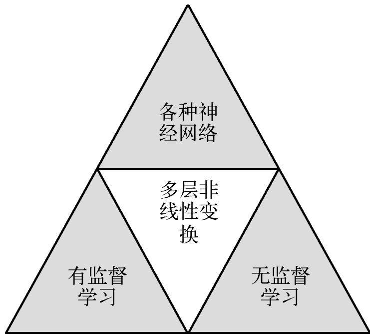


图 1.1　深度学习金字塔

1）有监督学习：训练数据包含了已知结果，根据这些结果来训练模型。

2）无监督学习：训练数据不含任何已知结果，在这种情况下，算法需要自身发现数据间的关系。

当使用 R 来实现深度学习时，采用的总体方法如图 1.2 所示。无论开发了什么特定的模型，最终还是会回到这个基本框架上来。数据输入到模型中，并且被多个非线性层过滤，最后一层由分类器构成，这个分类器决定了感兴趣的目标所属的类别。

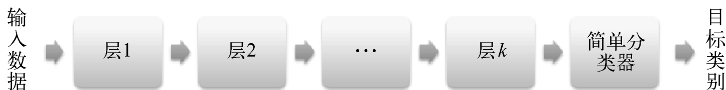


图 1.2　深度学习总体框架

深度学习的目的在于预测一个响应变量（分类变量）。这在某种程度上类似于线性回归做的事情。在线性回归中，一个线性模型使用一组独立变量（亦称属性或者特征）来预测响应变量。尽管如此，传统的线性回归模型并不被认为是深度的。

其他流行的学习方法，如决策树、随机森林以及支持向量机，都不是深度结构。决策树和随机森林使用的都是原始输入数据，对这些数据它们不做变换也不产生新的特征；支持向量是由一个核和一个线性变换构成的模型。

1.1.3　深度学习本质

深度学习能够以适当数目、并行、非线性步骤对非线性数据进行分类或者预测，它的威力正是源于此。从原始输入数据一直到数据分类结果，一个深度学习模型对输入数据进行分层次的学习，每一层从前一层的输出中提取特征。

深度学习模型是多隐藏层的神经网络。最简单的深度神经网络如图 1.3 所示，它包含了至少两个隐藏层。在这个神经网络中，每一层都把前一层的输出看成是本层的输入。

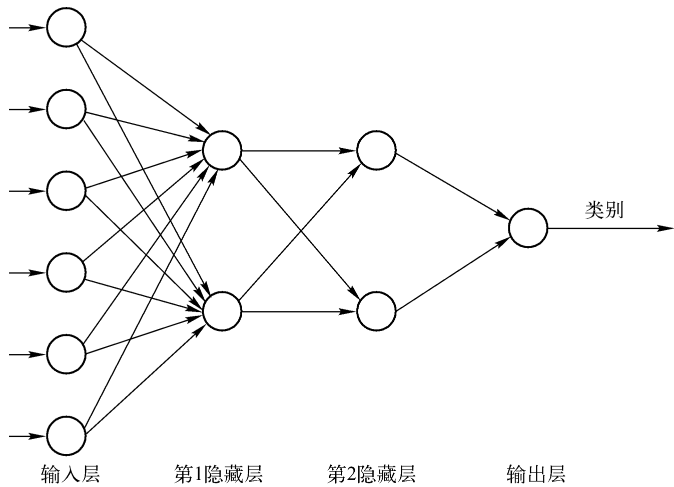


图 1.3　两个隐藏层的前馈神经网络

深度多层神经网络包含了许多非线性变换，这些变换使得深度多层神经网络能够简洁地近似表达复杂的非线性函数。深度多层神经网络善于分辨出数据中的复杂模式，它们已经被用来解决实际问题，如计算机视觉、自然语言处理、语音识别等。

1.1.4 深度学习应用

深度学习技术已经商业化了，应用于健康护理产业和医学图像处理，自然语言处理已经被用于广告业来改善点击率。微软、谷歌、IBM、雅虎、推特、百度、Paypal 以及 Facebook 都正在利用深度学习来理解用户的行为，目的是能够有针对性地推荐服务与产品。深度学习技术处处存在，很难想出在商业活动的哪个领域不需要利用深度学习，以下列出深度学习商业化的一些领域：

1）过程建模与控制。

2）健康诊断。

3）投资证券管理。

4）军事目标识别。

5）核磁共振图像与 X 射线图像分析。

6）银行以及其他财务机构对个人的信用评价。

7）市场推广。

8）语音识别。

9）股票市场预测。

10）文本搜索。

11）财务欺诈检测。

12）光学字符识别。

1.2 前向反馈神经网络 FNN

1.2.1 多层感知器

前向反馈神经网络（Feed－forward Neural Networks，FNN）的引入显著改善了预测的准确率，并且随着 FNN 训练方法的逐渐完善使得商业和科学研究持续受益。

对模拟人脑的生理结构以及功能的需求催生了 FNN。尽管这样的需求从来没有具体化，但人们很快发现 FNN 在分类和预测任务中表现得相当好。

FNN 能被用来解决许多分类问题。这是因为从理论上讲，它能逼近任何可以计

算的函数。在实践中，有些问题对错误比较宽容，有些问题不能简单地应用严格的规则，神经网络在应对这些问题时特别有效。

一个 FNN 是由一些相互连接的神经元构建而成。这些神经元通常被安置到一些层中。一个典型的前馈神经网络至少有一个输入层、一个隐藏层和一个输出层。输入层的神经元数对应于输入神经网络的特征或者属性的数目。这类似于在线性回归模型中使用的自变量。输出神经元的数目对应于分类数或者预测的类别数。隐藏层结点大致上是用来执行对原始输入特征的非线性变换。

最简单形式的前馈神经网络通过网络来传播特征信息，用来做出预测。前馈神经网络的输出要么是连续的（回归），要么是离散的（分类）。图 1.4 说明了一种典型的前馈神经网络拓扑结构。它有 7 个输入神经元、一个具有 5 个神经元的隐藏层，以及 3 个输出神经元。信息从输入特征前馈到隐藏层，并且接着再前馈到做出分类或者回归预测的输出层。它之所以叫前馈神经网络是因为信息是向前流动通过整个网络的。

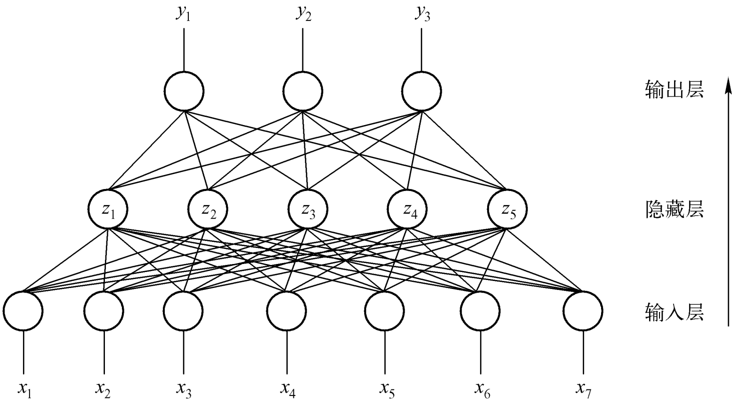


图 1.4　一种基本的 FNN

1.2.2　神经元的作用

图 1.5 说明了一个生物神经元是如何工作的。生物神经元通过电信号在相互之间传递信号或者消息。相邻的神经元通过它们的树突接收这些信号。信息从树突流向称为胞体的主细胞体，再通过轴突到轴突末端。实质上，生物神经元就是彼此之间相互传递与各种生物功能相关信息的计算机器。

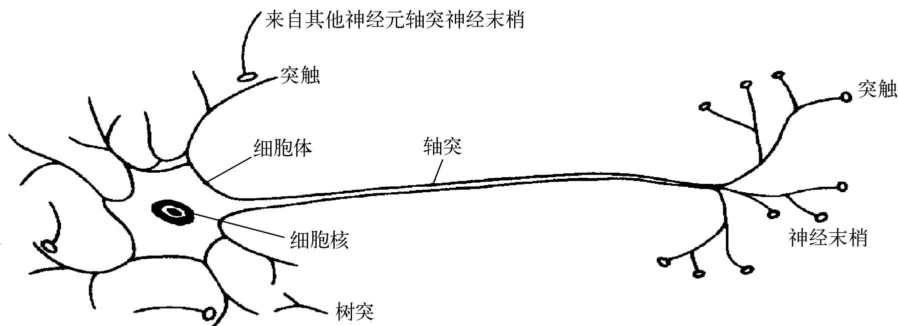


图 1.5　生物神经元

神经元是人工神经网络的核心，它是最基本的处理要素。输入层神经元接收进入网络的信息，这些信息通过一个数学函数来处理，接着被传递到隐藏层神经元。这种信息经过隐藏层神经元处理传递到输出层神经元。信息通过一个激活函数进行处理，这是神经元工作的关键。激活函数模拟了大脑神经元，大脑神经元是否工作取决于输入信号的强度。

第一个感知器模型是 1958 年在康奈尔航空实验室中开发的。它由无反馈的三层所构成：

1）传递输入数据到第二层（视网膜）。

2）把加权输入和阈值阶梯函数组合起来（神经元连接）。

3）输出层。

处理的结果接着被加权并且被传递到下一层的神经元中。实质上，神经元通过加权和相互激活，根据处理信息的权重来确定两个神经元之间的联系强度大小。

每一个神经元都包含一个激活函数和一个阈值。阈值是输入要激活神经元所必须的最小值。因此，神经元的任务就是，在传递输出到下一层之前，求输入信号的加权和，以及执行一个激活函数。

综上，输入层对输入数据进行求和；中间层神经元对由输入层神经元传输而来的加权信息进行求和；并且输出层对由中间层神经元传输而来的加权信息进行求和。

图 1.6 说明了神经元是如何工作的。给定一个输入样本的特征 $\{x_1,\cdots,x_n\}$ 和一个权重 W_i；接着计算神经元输入的加权和，公式如下：

$$y=\sigma(\text{net})=\sigma\left(\sum_{i=1}^{k}(W_i x_i+b_i)\right)$$

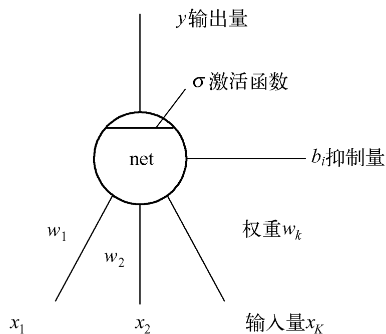


图 1.6　神经元的替代表示

参数 b_i 被称为抑制量（偏置），它类似于线性回归模型中的截距。它使得网络能够“上移”或者“下移”激活函数。这种弹性对于成功的机器学习是重要的。

1.2.3　激活函数

隐藏层结点需要激活函数实现非线性变换。一个神经元先执行激活函数，接着函数的输出传输到网络的下一层神经元。设计激活函数是为了限制神经元的输出，输出值通常被限定在 0 ~ 1 或者 -1 ~ 1 之间。在大部分情况下，一个网络中的每一个神经元的激活函数都是一样的。几乎所有的非线性函数都可以充当激活函数，尽管如此，对前向传播算法而言，激活函数必须是可微的，如果是有界函数，将会更有帮助。

双曲函数是一个常规的选择，这个函数是一个形如“S”的可微激活函数。双曲函数如图 1.7 所示，这里的 c 是一个值为 0 ~ 1 的常量，双曲函数很受欢迎，部分原因是求导容易，减少了训练时的代价。双曲函数的输出值也是在 0 ~ 1 之间，其表达式如下：

$$\sigma(x) = \frac{1}{1 + \exp(-cx)}$$

激活函数的数量很多，如

1）线性函数：

$$\sigma(x) = ax + b$$

2）双曲正切函数：双曲正切函数的输出范围是 -1 ~ 1。这个函数与双曲函数有许多共同的特性；但由于其输出范围更大，有时候在对非线性关系建模时，其会更有效。它的函数表达式如下：

$$\sigma(x) = \tanh(cx)$$

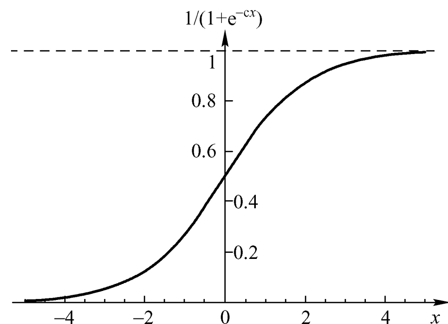


图 1.7　双曲函数

3）Softmax 函数（Logistic – sigmoid、Tanh – sigmoid）：神经网络使用一个 Softmax 函数来得到输出是很常见的。使用这个函数可以得到 k 个类别上的概率分布。其函数表达式如下：

$$\sigma(x)=\frac{\exp\left(\frac{x}{T}\right)}{\sum_{k}\exp\left(\frac{x}{T}\right)}$$

式中，T 是温度（通常设定为 1），它通常和一个具有 k 个类别的响应变量一起使用。基本上，它可以被看成一系列的二值结点，这些结点相互约束以便使 k 个状态中的某一个值为 1，同时其余的状态值为 0。

4）矫正线性函数（ReLU）：它的表达式如下：

$$\sigma(x)=\max(0,x)$$

这个激活函数在深度学习模型中很常用，因为对于语音识别和计算机视觉任务而言，它显著改善了分类速度。只有在神经元的输出值是正数时，它才允许激活神经元。这个激活函数还能够使其所在网络的计算速度快于采用双曲激活函数或者双曲正切激活函数的网络，这是因为它只是一个简单的求取最大值的操作。矫正激活函数使神经网络稀疏，因为当随机初始化时，整个网络中的神经元大约有一半被设置成 0。

还有一种平滑估计，表达式如下：

$$\sigma(x)=\log(1+\exp(x))$$

1.2.4　学习算法

学习算法有很多种，一般而言，它们都是通过迭代修改网络权重，直到网络输出与期望输出的误差低于一个预先设定的阈值。

反向传播算法是最流行的学习算法，并且现在还在广泛使用。它将梯度下降法作为核心的学习机制。从随机权重开始，反向传播算法计算权重值是根据网络输出与期望输出之间的误差逐渐调整权重值。

这个算法将误差从输出端传播到输入端，并且使用梯度下降逐渐微调网络权重使误差值最小化。

学习步骤如下：

1）定义变量与参数：$\boldsymbol{x}$（输入向量）、$\boldsymbol{w}$（权值向量）、b（偏置）、y（实际输出）、$\hat{y}$（期望输出）和 α（学习率参数）。

2）初始化：$n=0$，$w=0$。

神经网络通过随机设定权重值和偏置进行初始化。首要规则是将随机值设定在一个范围内（$-2n\sim 2n$），这里 n 是输入特征的个数。

3）输入训练样本：对每个训练样本指定其期望输出：A 类记为 1；B 类记为 −1。

4）前馈：在网络中，信息从输入层到隐藏层再到输出层的向前传递，并且传递过程要借助激活函数和加权，计算实际输出 $y=\boldsymbol{\sigma}(\boldsymbol{wx}+b)$。

5）更新权值向量：利用梯度调整权重，以达到减小误差的目的。

$$w(n+1)=w(n)+\alpha[\hat{y}-y(n)]x(n)$$

每个神经元的权重和偏置都是按照一定规则调整的，这个规则依赖于激活函数的导数、网络输出与实际目标结果之间的差别以及神经元的输出。

图 1.8 粗略地说明了权重调整的基本思想。如果偏导数是负的，要增加权重（见图 1.8a）；如果偏导数是正的，要减少权重（见图 1.8b）。这种学习流程的每一个循环，称为迭代（Epoch）。

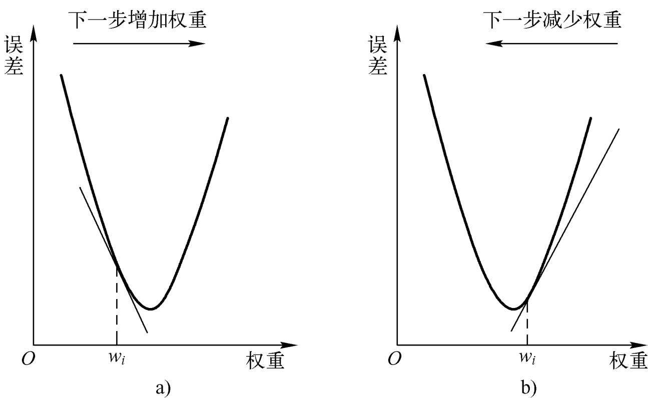


图 1.8　反向传播算法的基本思想

6）误差评估：将网络输出与已知输出进行比较，以此评估网络权重。如果误差低于预先设定的阈值，网络训练完毕，并且算法终止，否则返回步骤3）。

早期使用梯度下降的反向传播算法经常收敛得很慢或者根本就不收敛。找到全局最优解，避免局部最小值，这是个挑战，因为，一个典型的神经网络可能有成千上万的权重，用这些权重的组合来求解。这个解经常是高度非线性的，这使得优化过程变得复杂。为了避免网络陷入局部最小，通常要指定一个动量参数。

1.3 R语言基础

1.3.1 入门

（1）什么是R语言

R语言是一个开源的数据分析环境，起初是由数位统计学家建立起来的，以更好地进行统计计算和绘图。由于R语言可以通过安装扩展包（Packages）而得到增强，所以其功能已经远远不限于统计分析。

R语言名称则是来源于两位主要作者的首字母（Robert Gentleman 和 Ross Ihaka）。

（2）为什么要学习R语言

1）免费开源。

现在很多学术期刊都对分析软件有版权要求，而免费的分析工具可以使你在这方面不会有什么担心。另外，如果学术界出现一种新的数据分析方法，那么要过很长一段时间才会出现在商业软件中。但开源软件的好处就在于，很快就会有人将这种方法编写成扩展包，或者你自己就可以做这份工作。

2）小巧而精悍。

R语言的安装包很小，大约40 MB，相比其他几个软件相比，它算是非常小巧精悍的。目前R语言非常受到专业人士欢迎，根据对数据挖掘大赛胜出者的调查可以发现，他们用的工具基本上都是R语言。此外，从最近几次R语言大会上可以了解到，咨询业、金融业、医药业都在大量的使用R语言，包括Google和Facebook的大公司都在用它。因此，学习R语言对职业发展一定是有帮助的。

3）丰富的R包。

R语言应该是所有数据分析软件里，方法（函数）最多的语言。截至2016年6

月，R 语言综合典藏网（Comprehensive R Archive Network，CRAN）上共提供了 3024 个包，涵盖了贝叶斯推断、分类方法、计量经济学、生态学、金融学、遗传学、机器学习、稳健统计、空间统计、生存分析和时间序列等多个方面。

4）广泛的数据接口。

R 语言可以良好地接入 CSV（Comma Separated Values）数据，或者通过其他包来扩展，直接读入 SPSS、SAS、Minitab、Stata 和 Excel 等文件，或者直接读取 MySQL、SQL Server、DB2 和 Oracle 等数据库。

5）强大的绘图功能。

R 语言提供了“高水平”“低水平”和“交互式”三种绘图命令，而且很容易生成 ps、pdf、png、jpeg、bmp、gif、SVG，甚至以 LATEX 或 HTML 形式输出。

（3）认识 R 语言环境

从图 1.9 可以看出，R 语言环境总共有四个工作区域，左上是用来写代码的，左下也可以写代码，同时也是数据输出的地方。R 语言是动态语言，写代码的形式有两种，一种是像写作文一样写很多，也就是像 C 语言一样的代码；另一种则是写一句就编译解释一句。左下就是写一句编译解释一句的工作区域（命令控制台）。右上是工作空间（Workspace）和历史记录。右下有四个主要的功能，Files 是查看当前 Workspace 下的文件，Plots 是展示运算结果输出的图形，Packages 能查看系统已安装的软件包，并且通过勾选载入内存，Help 则是查看帮助文档。

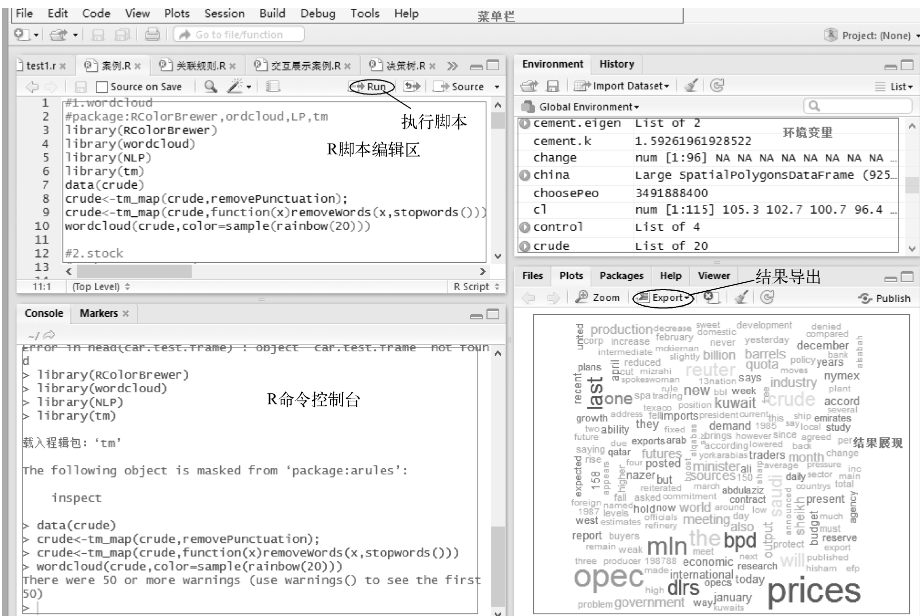


图 1.9　Rstudio 界面

1.3.2 基本语法

1）赋值：<– 也可用 = ，甚至 –> 代替。

c <– a + b，c = a + b，a + b –> c 等价

2）注释：#ABC，无多行注释。

3）变量：变量无须定义，但区分大小写，注意 China 和 china 的不同。

4）获取帮助，具体函数见表 1.1。

表 1.1 获取帮助

函　　数	功　　能
help. start( )	打开帮助文档首页
help("foo") 或？foo	查看函数 foo 的帮助（引号可以省略）
help. search("foo") 或？？foo	以 foo 为关键词搜索本地帮助文档
example("foo")	函数 foo 的使用示例（引号可以省略）
apropos("foo" ,mode = "function")	列出名称中含有 foo 的所有可用函数
data( )	列出当前已加载包中所含的所有可用示例数据集
vignette( )	列出当前已安装包中所有可用的 vignette 文档
vignette("foo")	为主题 foo 显示指定的 vignette 文档

5）工作空间：工作空间是 R 语言用来读取文件和保存结果的默认目录（见表 1.2）。

表 1.2 用于管理 R 语言工作空间的函数

函　　数	功　　能
getwd( )	显示当前的工作目录
setwd("mydirectory")	修改当前的工作目录为 mydirectory
ls( )	列出当前工作空间中的对象
rm(objectlist)	移除（删除）一个或多个对象
options( )	显示或设置当前选项
q( )	退出 R，将会询问你是否保存工作空间

6）包：包是函数、数据、预编译代码以一种定义完善的格式组成的集合。

① libpath( )　　　　　　　#显示包所在位置

② library( )　　　　　　　　#显示当前工作空间有哪些包

③ install. packages("包名")　　　　#安装包

④ library("包名")　　　　　　#加载包到内存

⑤ help(package ="包名")　　　　#输出包的简短描述以及包中函数和数据集名称的列表

7）内存管理：

gc(rm(list = ls( )))　　　　　　#清除内存所有变量

memory. limit(1000000)　　　　　#设置内存上限

1.3.3　数据

（1）数据集

不同的行业对于数据集的行和列叫法不同。统计学家称它们为观测（Observation）和变量（Variable），数据库分析师则称其为记录（Record）和字段（Field），数据挖掘/机器学习学科的研究者则把它们叫作示例（Example）和属性（Attribute）。在本书中通篇使用术语观测和变量。

从表 1.3 可以清楚地看到此数据集的结构（本例中是一个数据框）以及其中包含的内容和数据类型。其中，PatientID 是实例标识符，AdmDate 是日期型变量，Age 是整型变量，Diabetes 是名义型变量，Status 是有序型变量。

表 1.3　病例数据

病人编号 (PatientID)	入院时间 (AdmDate)	年龄 (Age)	糖尿病类型 (Diabetes)	病情 (Status)
1	10/15/2009	25	Type1	Poor
2	11/01/2009	34	Type2	Improved
3	10/21/2009	28	Type1	Excellent
4	10/28/2009	52	Type1	Poor

R 语言提供了许多数据对象，包括标量、向量、数组、数据框和列表。多样化的数据对象赋予了 R 语言极其灵活的数据处理能力。

R 语言可以处理的数据类型（模式）包括数值型、字符型、逻辑型（TRUE/FALSE）、复数型（虚数）和原生型（字节）。在表 1.3 中，PatientID、AdmDate 和 Age 为数值型变量，而 Diabetes 和 Status 为字符型变量。另外，需要分别告诉 R，Pa-

tientID 是实例标识符，AdmDate 是日期数据，Diabetes 和 Status 分别是名义型和有序型变量。R 语言将实例标识符称为行名（Rownames），将类别（包括名义型和有序型）变量称为因子（Factors），或称为响应变量、决策变量，或类别变量。

（2）数据对象

R 语言提供了图 1.10 所示的数据对象。

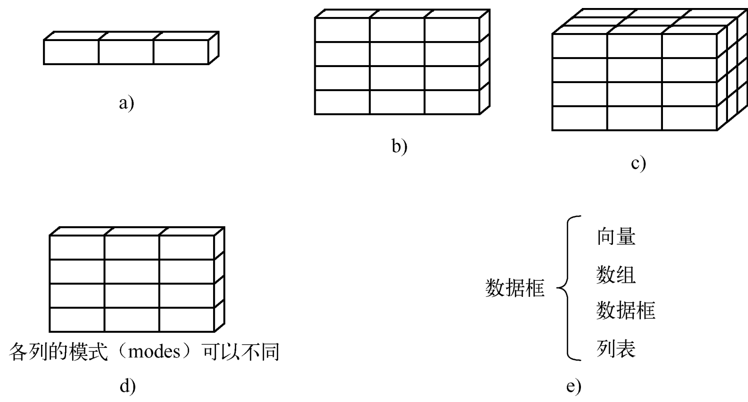


图 1.10　R 语言中的数据对象

a）向量　b）矩阵　c）数组　d）数据框　e）数据框

1）向量。

向量是用于存储数值型、字符型或逻辑型数据的一维数组。函数 c( ) 可用来创建向量。如：

```
a <- c(1,2,3,4)
b <- c("US","CHINA","ENGLISH")
c <- c(TRUE,TRUE,FALSE)
```

这里，a 是数值型向量；b 是字符型向量；而 c 是逻辑型向量。

说明 1：单个向量中的数据必须拥有相同的类型或模式（数值型、字符型或逻辑型）。同一向量中无法混杂不同模式的数据。

说明 2：标量是只含一个元素的向量，例如 f <- 3、g <- " US" 和 h <- TRUE。它们用于保存常量。

通过在方括号中给定元素所处位置的数值，可以访问向量中的元素。例如，a[c(2, 4)]用于访问向量 a 中的第二个和第四个元素。更多示例如下：

```
a[2:4]      #访问第 2,3,4 个元素
```

```
a[3]          #访问第 3 个元素
```

2）矩阵。

矩阵是一个二维数组，只是每个元素都拥有相同的类型（数值型、字符型或逻辑型）。可通过函数 matrix 创建矩阵。一般使用格式如下：

```
myymatrix <- matrix(vector,nrow = number_of_rows,ncol = number_of_columns,
                    byrow = logical_value,dimnames = list(
                    char_vector_rownames,char_vector_colnames))
```

其中 vector 包含了矩阵的元素，nrow 和 ncol 用以指定行和列的维数，dimnames 包含了可选的、以字符型向量表示的行名和列名。选项 byrow 则表明矩阵应当按行填充（byrow = TRUE）还是按列填充（byrow = FALSE），默认情况下按列填充。

【例 1.1】 定义 5 ×4 矩阵的两种方式。

方法 1：

```
y <- matrix(1:20,nrow =5,ncol =4)
      [,1] [,2] [,3]  [,4]
[1,]    1    6   11    16
[2,]    2    7   12    17
[3,]    3    8   13    18
[4,]    4    9   14    19
[5,]    5   10   15    20
```

方法 2：

```
cells <- c(1:20)
rnames <- c("R1","R2","R3","R4","R5")
cnames <- c("C1","C2","C3","C4")
y <- matrix(cells,nrow =2,ncol  =  2,byrow =TRUE, list(rnames,cnames))
y
   C1 C2 C3 C4
R1  1  6 11 16
R2  2  7 12 17
R3  3  8 13 18
R4  4  9 14 19
R5  5 10 15 20
```

可以使用下标和方括号来选择矩阵中的行、列或元素。

y[i,]返回矩阵 ***y*** 中的第 i 行；

y[,j]返回矩阵 ***y*** 第 j 列；

y[i,j]返回矩阵 ***y*** 第 i 行第 j 列元素。

选择多行或多列时，下标 i 和 j 可为数值型向量，如 y[c(1,3),c(2:4)]。

注意：R 语言中的下标不是从 0 开始，而是从 1 开始。

3）数组。

数组（Array）与矩阵类似，但是维度可以大于 2。数组可通过 array 函数创建，形式如下：

```
myarray <- array(vector,dimentins,dimnames)
```

其中 vector 包含了数组中的数据，dimensions 是一个数值型向量，给出了各个维度下标的最大值，而 dimnames 是可选的、各维度名称标签的列表。

如下代码给出了一个创建三维（2×3×4）数值型数组的示例：

```
z <- array(1:24,c(2,3,4))
```

数组是矩阵的一个自然推广。它们在编写新的统计方法时可能很有用。像矩阵一样，数组中的数据也只能是同类型。

从数组中选取元素的方式与矩阵相同。例如，元素 z[1,2,3]为 15。

4）数据框。

由于不同的列可以包含不同类型（数值型、字符型等）的数据，数据框的概念较矩阵来说更为一般。数据框是在 R 语言中最常处理的数据对象。

表 1.3 所示的病例数据集包含了数值型和字符型数据。由于数据有多种类型，无法将此数据集放入一个矩阵。在这种情况下，使用数据框是最佳选择。

数据框可通过函数 data.frame()创建：

```
x <- data.frame(col1, col2, col3,…)
```

其中的列向量 col1，col2，col3,… 可为任何类型（如字符型、数值型或逻辑型）。

数据框的引用与矩阵一样，例如，x[1:3,],x[2,c(2,4)]，除此之外，增加通过变量引用数据框元素的方法，如把表 1.3 的数据框定义为 x，则 x$age 等价于 x[,3]。

5）因子。

变量可归结为名义型、有序型或连续型变量。名义型变量是指没有顺序之分的类别变量。例如，糖尿病类型 Diabetes(Type1,Type2)是名义型变量的一例。即使在

数据中 Type1 编码为 1，而 Type2 编码为 2，这也并不意味着二者是有序的。有序型变量表示的是一种顺序关系，而非数量关系。例如，病情 Status(poor，improved，excellent）是顺序型变量的典型示例。我们知道，病情为 Poor（较差）病人的状态不如 Improved（病情好转）的病人，但并不知道相差多少。连续型变量可以呈现为某个范围内的任意值，并同时表示了顺序和数量。年龄 Age 就是一个连续型变量，它能够表示像 14.5 或 22.8 这样的值以及其取值范围内的其他任意值。

类别变量（名义型变量、响应变量）和有序类别（有序型）变量在 R 语言中称为因子（Factor)。因子在 R 语言中非常重要，因为它决定了数据的分析方式以及如何进行视觉呈现。

函数 factor( ) 以一个整数向量的形式存储类别值，整数的取值范围是［1… k ］(其中 k 是变量中唯一值的个数)，同时一个由字符串（原始值）组成的内部向量将映射到这些整数上。

举例来说，假设有向量：

```
diabetes <- c("type1","type2", "type1", "type1")
```

语句 diabetes <- factor(diabetes) 将此向量存储为（1,2,1,1)，并在内部将其关联为 1 = Type1 和 2 = Type2（具体赋值根据字母顺序而定)。针对向量 diabetes 进行的任何分析都会将其作为名义型变量对待，并自动选择适合这一测量尺度的统计方法。

要表示有序型变量，需要为函数 factor( ) 指定参数 ordered = TRUE。给定向量：

```
status <- c("Poor", "Improved","Excellent", "Poor")
```

语句 status <- factor(status,ordered = TRUE) 会将向量编码为（3,2,1,3)，并在内部将这些值关联为 1 = Excellent、2 = Improved 以及 3 = Poor。

对于字符型向量，因子的水平默认依字母顺序创建，这对于因子 status 是有意义的，因为“Excellent”“Improved”“Poor”的排序方式恰好与逻辑顺序相一致。如果“Poor”被编码为“Ailing”，会有问题，因为顺序将为“Ailing”“Excellent”“Improved”。如果理想中的顺序是“Poor”“Improved”“Excellent”，则会出现类似的问题。按默认的字母顺序排序的因子很少能够让人满意。

可以通过指定 levels 选项来覆盖默认排序。例如：

```
status <- factor(status,order = TRUE,levels = c("Poor","Improved", "Excellent"))
```

各水平的赋值将为 1 = Poor、2 = Improved、3 = Excellent。需保证指定的水平与数据中的真实值相匹配，因为任何在数据中出现而未在参数中列举的数据都将被设为

默认值。

6）列表。

列表（List）是 R 语言的数据类型中最为复杂的一种。一般来说，列表就是一些对象或成分（Component）的有序集合。列表允许整合若干（可能无关的）对象到单个对象名下。例如，某个列表中可能是若干向量、矩阵、数据框，甚至其他列表的组合。可以使用函数 list( )创建列表：

```
mylist <- list(object1, object1,…)
```

其中的 objecti 可以是目前为止讲到的任何结构。

【例 1.2】

```
g <- "my first list"                    #定义字符串
h <- c(26,26,18,29)                     #定义长度为 4 的数值向量
j <- matrix(1:10,nrow =5)               #定义 5 ×2 矩阵
k <- data.frame(c(1,2),c(3,4))          #定义数据框
mylist <- list(title = g,ages = h,j,k)  #定义列表
mylist
```

也可以通过在双重方括号中指明代表某个成分的数字或名称来访问列表中的元素。此例中，mylist[[2]]和 mylist[["ages"]]均指那个含有四个元素的向量。由于以下两个原因，列表成为 R 中的重要数据结构：首先，列表允许以一种简单的方式组织和重新调用不相干的信息；其次，许多 R 函数的运行结果都是以列表的形式返回的，需要取出其中哪些成分由分析人员决定。

1.3.4 绘图

（1）图形参数

par( )返回当前绘图参数的列表。常用绘图参数列于表 1.4 中。

表 1.4 常用绘图参数

类　　型	参　　数	用　　途	值
符号与线条	pch	绘制点时的符号形状	0 ~ 25
	cex	符号的大小	默认大小的倍数
	lty	线条的类型	1 ~ 6
	lwd	线条的宽度	默认大小的倍数

（续）

类　型	参　数	用　途	值
颜色	col	绘图颜色	字符向量
	col. axis	坐标轴刻度	
	col. lab	坐标轴标签	
	col. main	标题	
	col. sub	副标题	
	fg	前景	
	bg	背景	

（2）自定义属性

1）标题 title( )。

2）坐标轴 axis( )。

3）参考线 abline(h = yvalues, v = xvalues)。

4）图例 legend(location, title, legend, …)，legend 是图例标签的字符向量。

5）文本标注 text( ), mtext( )。

（3）图形组合

1）par(mfrow = c(nrows, ncols))。

2）layout(matrix)，matrix 指定了图形组合的位置布局。

（4）基本图形

1）条形图：barplot(x, main = " ", xlab = " ", ylab = " ", horiz = FALSE)。

2）饼状图：pie( )。

3）扇形图：plotrix 包的 fan. plot(x, labels = c( ))，适合于大小的比较。

4）直方图：hist(x, breaks = n, …)，breaks 可以指定直方图的条数。

5）核密度图：密度函数 density(x)，密度图的绘制可以是 plot(density( )) 直接绘制，或者是在一幅图上面通过 lines(density( )) 进行叠加，注意：在叠加之前由于密度图都小于 1，因此要把之前的图范围变为 1 处理，添加属性 freq = FALSE。

6）箱线图：boxplot( )；箱线图分组展示：boxplot(formula, data = dataframe)，formula 形如 Y – X，为变量 X 的每个值生成对应 Y 的箱线图。

7）点图：dotchart( )。

8）散点图：plot( )。

9）回归曲线：lm( )。

10）平滑的曲线：lowess(x,y)。

表 1.5 中列出了 R 语言中部分用于数据展现的图形。

表 1.5　R 中部分用于数据展现的图形

要表达的数据和信息	建议采用图形					
	饼图	垂直柱	水平柱	线图	水泡	其他
整体的一部分						
不同数据的比较						
时间序列						
频率						
两组数据的相关性						
和多重数据、标准相比较						

1.3.5　数据准备

（1）数据导入

对初学者来讲，面对一片空白的命令行窗口，第一个任务就是数据的导入。数据导入有很多途径，例如从网页抓取、公共数据源获得、文本文件导入。

例如，读取 iris.csv 演示数据，在 R 语言中输入如下命令：

```
data <- read.table('iris.csv',T)
```

或

```
data <- read.csv('iris.csv',T)
```

这里的 read. table(read. csv)是 R 语言读取外部数据的常用命令，T 表示第一行是表头信息，整个数据存在名为 data 的变量中。另一种更方便的导入方法是利用 Rstudio 的功能，在 Workspace 菜单选择“import dataset”也是一样的。

如果导入的文件大小超过 150 MB，属于大文件，要使用以下命令：

```
library(data. table)
c1 <- fread("abc. txt",encoding = "UTF - 8",sep = "\t",data. table = FALSE)
```

参数 data. table = FALSE 表示导入的结果为数据框，否则为 data. table 类型，默认为 data. table 类型。

（2）数据子集

如果只关注数据的一部分，例如从原数据中抽取第 20 ~ 30 号样本的 Sepal. Width 变量数据，因为 Sepal. Width 变量是第 2 个变量，所以此时键入下面的命令即可：

```
newdata <- data[20:30,2]
```

如果需要抽取所有数据的 Sepal. Width 变量，那么下面两个命令是等价的：

```
newdata <- data[ ,2]
newdata <- data$Sepal. Width
```

（3）数据描述

str(x):显示数据 x 的结构，如变量数量、变量类型、观测值等。

summary(patient)：显示统计概要。

dim(patientdata)：显示数据框 patientdata 维数。

（4）抽样函数

如果从 1 ~ 10 中随机抽取 5 个数字，则首先产生一个序列，然后用 sample 函数进行无放回抽取。

```
x = 1:10
sample(x,size = 5)
```

有放回抽取则是

```
sample(x,size = 5,replace = T)
```

sample 函数在建模中经常用来对样本数据进行随机的划分，一部分作为训练数据；另一部分作为检验数据。

1.3.6 基本运算

（1）常用计算函数

1）mean(x)：均值。

2）sum(x)：求和。

3）min(x)：最小值。

4）max(x)：最大值。

5）var(x)：方差。

6）sd(x)：标准差。

7）cov(x,y)：协方差。

8）cor(x,y)：相关度。

9）prod(x)：所有值相乘的积。

10）which(x 的表达式)：如 which. min(x)，which. max(x)。

11）rev(x)：反转。

12）sort(x)：排序。

（2）行列命名

1）colnames(matrix) = c("","",…)。

2）rownames(matrix) = c("","",…)。

（3）矩阵运算

1）矩阵相乘：A% * % B。

2）t(matrix)：矩阵转置。

3）diag(matrix)：矩阵的对角（向量）；diag(diag(matrix))，对角矩阵。

4）solve(matrix)：矩阵求逆。

5）eigen(matrix)：特征值和特征向量。

6）svd(matrix)：奇异值分解，返回 X 包含属性 U、d、V。

1.4 FNN 的 R 实现

【例 1.3】以鸢尾花数据集（iris）为例，将鸢尾花数据分为两类：setosa、versicolor。输入特征为花瓣长度与宽度。不使用包编写感知器 R 脚本。

（1）iris 数据结构

```
> str(iris)
' data. frame'  :150 obs. of 5 variables:
 $Sepal. Length :num 5. 1 4. 9 4. 7 4. 6 5 5. 4 4. 6 5 4. 4 4. 9 ...
 $Sepal. Width  :num 3. 5 3 3. 2 3. 1 3. 6 3. 9 3. 4 3. 4 2. 9 3. 1...
 $Petal. Length :num 1. 4 1. 4 1. 3 1. 5 1. 4 1. 7 1. 4 1. 5 1. 4 1. 5 ...
 $Petal. Width  :num 0. 2 0. 2 0. 2 0. 2 0. 2 0. 4 0. 3 0. 2 0. 2 0. 1...
 $Species       :Factor w/3 levels "setosa","versicolor",...:1 1 1 1
1 1 1 1 1 1...
```

（2）建模

```
> a <- 0.2                                    #学习率
> w <- rep(0,3)                               #初始权重
> iris1 <- t(as.matrix(iris[,3:4]))           #训练数据
> d <- c(rep(0,50),rep(1,100))                #理论输出值
> e <- rep(0,150)
> p <- rbind(rep(1,150),iris1)                #给数据加标签
> max <- 100000                               #迭代次数上限
> eps <- rep(0,100000)                        #初始误差
> i <- 0                                      #初始化迭代次数
> repeat{                                     #开始迭代
    v <- w%*%p;                               #计算加权和
    y <- ifelse(sign(v) >= 0,1,0);            #根据激活函数计算实际输出值
    e <- d - y;                               #计算误差
    eps[i+1] <- sum(abs(e))/length(e)         #结算平均误差
    if(eps[i+1] < 0.01){                      #达到逼近精度,则迭代终止
        print("finish:");
        print(w);
        break;
    }
    w <- w + a*(d-y)%*%t(p);                  #修改权重
    i <- i + 1;                               #迭代次数
    if(i > max){                              #如果达到迭代次数上限,则迭代终止
        print("max time loop");
        print(eps[i])
```

```
            print(y);
            break;
        }
}
[1] "finish:"
            Petal. Length Petal. Width
[1,] -39.6          10.82          18.82
```

(3) 可视化

```
#分类结果如图 1.11 所示
>plot(Petal.Length ~ Petal.Width,xlim = c(0,3),ylim = c(0,8),
data = iris[iris$Species == "virginica",])
>data1 <- iris[iris$Species == "versicolor",]
>points(data1$Petal.Width,data1$Petal.Length,col = 2)
>data2 <- iris[iris$Species == "setosa",]
>points(data2$Petal.Width,data2$Petal.Length,col = 3)
>x <- seq(0,3,0.01)
>y <- x * ( -w[2]/w[3]) - w[1]/w[3]
>lines(x,y,col = 4)
```

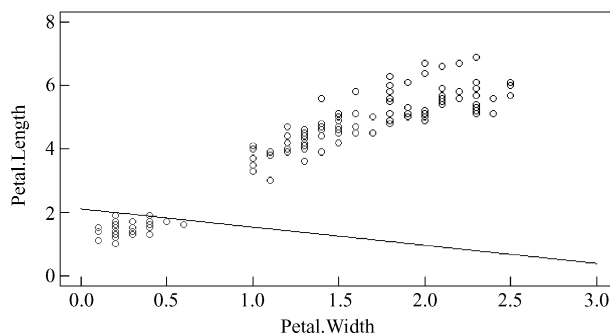


图 1.11　分类结果图

这是运行了 7 次得到的结果。感兴趣的读者可以与支持向量机相比，表明神经网络的单层感知器分类不是那么的可信，有些弱。可以发现交叉验证结果并不理想。

```
>plot(1:i,eps[1:i],type = "o")   #每次迭代的平均绝对误差,如图 1.12 所示
```

【例 1.4】 例 1.3 续

使用 nnet 包创建单层 FNN 模型。

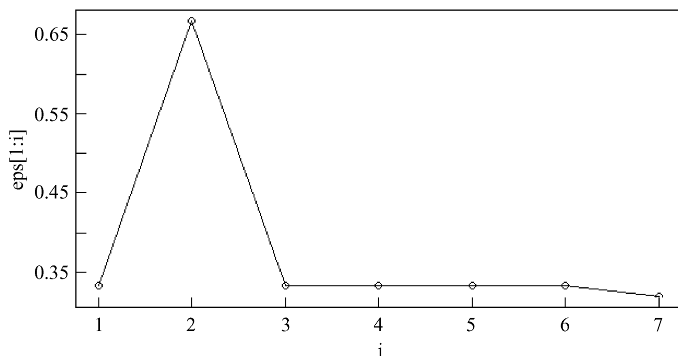


图 1. 12　每次迭代的平均绝对误差

（1）加载使用的包和数据

```
>library(nnet);                #安装 nnet 软件包
>library(mlbench);             #安装 mlbench 软件包
```

（2）数据准备

随机选择半数观测作为训练集，剩下的作为测试集。

```
>set.seed(1);                  #设随机数种子
>ir <- rbind(iris3[,1],iris3[,2],iris3[,3])
>targets <- class.ind(c(rep("s",50),rep("c",50),rep("v",50)))
>samp <- c(sample(1:50,25),sample(51:100,25),sample(101:150,25))
```

（3）建模

建模用到 nnet 包中的 nnet 函数，其调用格式为

```
>ir1 <- nnet(ir[samp,],                #数据集
    targets[samp,],
    size =2,                           #隐层结点数
    rang =0.1,
    softmax = censored,                #分类算法
    decay =5e -4,                      #表明权值是递减的(可以防止过拟合)
    maxit =200                         #最大迭代次数
    )
```

构建只包含有 3 个结点的一个隐藏层神经网络。

（4）模型部署

适用于神经网络的部署方法为 predict。

```
test.cl <- function(true,pred) {
    true <- max.col(true)
    cres <- max.col(pred)
    table(true,cres)
}
test.cl(targets[ -samp, ],predict(ir1,ir[ -samp, ]))
```

true \ cres	1	2	3
1	24	0	1
2	0	25	0
3	7	0	18

1.5 学习指南

FNN 简称神经网络，是一种模仿生物神经网络的结构和功能的数学模型或计算模型。FNN 由大量的人工神经元连接在一起进行计算。大多数情况下 FNN 能在外界信息的基础上改变内部结构，是一种自适应系统。现代 FNN 是一种非线性统计性数据建模工具，常用来对输入和输出间复杂的关系进行建模，或用来探索数据的模式。

FNN 从以下四个方面去模拟人的智能行为：

1）物理结构：神经元模拟生物神经元的功能。

2）计算模拟：人脑的神经元有局部计算和存储的功能，通过连接构成一个系统。FNN 中也有大量有局部处理能力的神经元，能够将信息进行大规模并行处理。

3）存储与操作：人脑和人工神经网络都是通过神经元的连接强度来实现记忆存储功能，同时为概括、类比、推广提供有力的支持。

4）训练：同人脑一样，FNN 将根据自己的结构特性，使用不同的训练、学习过程，自动从实践中获得相关知识。

FNN 是一种运算模型，由大量的结点（或称“神经元”或“单元”）之间相互连接构成。每个结点代表一种特定的输出函数，称为激励函数。每两个结点间的连接代表通过该连接的信号加权值，称为权重，这相当于人工神经网络的记忆。网络的输出则依据网络的连接方式、权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

第 2 章　深度神经网络 DNN

在实际应用中，深度学习技术已经取得了非凡的成就。经过几十年的研究，几乎所有的，尤其是核心的那几位研究者都认为这些应用以前是不可能的，现在都变得可行了。因为相比于其他先进的机器学习工具，深度神经网络（Deep Neural Network，DNN）在一系列诸如目标检测、场景理解、遮挡检测等应用中表现出了优越的性能。

DNN 模型在包括过拟合和鲁棒性方面都具有优秀的预测能力，因此它得到了快速的发展（商业领域）和广泛的应用，并且它的应用还在逐渐增长。

本章将讨论几个在 DNN 上应用的例子：雾天视觉增强、恶意软件检测、图像压缩和函数逼近。通过这些例子的学习，将学会使用一系列 R 包，掌握一些数据科学技巧，获得一些提高 DNN 性能的建议。

2.1　DNN 原理

线性模型通过特征间的线性组合来表达“结果 - 特征集合”之间的对应关系。由于线性模型的表达能力有限，在实践中，只能通过增加“特征计算”的复杂度来优化模型。比如，在广告点击通过率（CTR）预估应用中，除了“标题长度、描述长度、位次、广告 ID，Cookie”等这样的简单原始特征，还有大量的组合特征（比如“位次 - Cookie”表示用户对位次的偏好）。事实上，现在很多搜索引擎的广告系统用的都是 Logistic Regression 模型（线性），而建模最重要的工作之一就是“特征抽取”。

线性模型的思路是“简单模型 + 复杂特征”，用这样的组合实现复杂非线性场景描述。由于模型结构简单，这种做法的训练/预估计算代价相对较小；但是，特征的选取是一个需要耗费大量人力的工作，且要求相关人员对业务有较深的理解。

建模的另外一个思路是“复杂模型 + 简单特征”。即弱化特征工程的重要性，利

用复杂的非线性模型来学习特征间的关系，增强表达能力。DNN 模型就是这样一个非线性模型。

如图 2. 1 所示，深度神经网络包括一个输入层，一个输出层，还有介于这两层之间的多个隐藏层。它类似于多层感知机，但是它拥有多个隐藏层，而且每个隐藏层都有多个相互连接的神经元。DNN 的多个隐藏层的优势在于能够逼近复杂的决策函数。

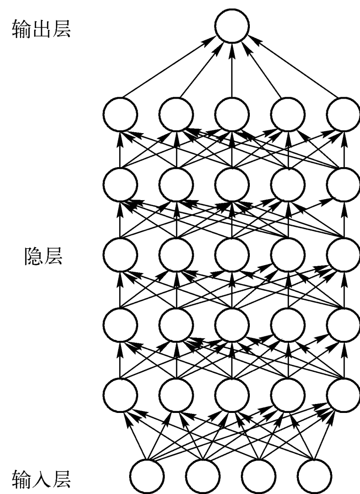


图 2. 1　DNN 模型

隐藏层的作用是实现复杂函数的转换。它们连接着输入层，为输入值标记权重并且结合对输入值标记的权重产生一个全新的、真实的数值，然后传递给输出层。输出层使用在隐藏层中计算得来的抽象的特征进行分类或者预测。

若想在很短时间内，全面地、明确地理解 DNN，隐藏层便是 DNN 的秘密所在，隐藏层神经元执行的非线性数据转换是该技术的核心。

从图 2. 1 可以看到，DNN 是神经元个体的组合，这些神经元相互连接，可以使输出的组合更加灵活。正是这样的灵活性，使得它们可以逼近任何函数。

多层的好处是可以用较少的参数表示复杂的函数。

在有监督学习中，以前的多层神经网络的问题是容易陷入局部极值点。如果训练样本足够充分覆盖未知样本，那么学到的多层权重可以很好地用来预测新的测试样本。但是很多任务难以得到足够多的标记样本，在这种情况下，简单的模型，如线性回归或者决策树往往能得到比多层神经网络更好的结果（更好的泛化性以及更小的训练误差）。

在无监督学习中，以往没有有效的方法构造多层网络。多层神经网络的顶层是

底层特征的高级表示，例如，对图像而言，底层是像素点，上一层的结点可能表示横线、三角；而顶层可能有一个结点表示人脸。一个成功的算法应该能让生成的顶层特征最大化地代表底层的样例。如果对所有层同时训练，时间复杂度会太高；如果每次训练一层，误差就会逐层传递。这会面临与上面有监督学习中相反的问题，会严重欠拟合。

2006 年，Hinton 提出了在无监督数据上建立多层神经网络的一种有效方法，简单地说，分为两步：一是每次训练一层网络；二是微调。微调的目的是使原始数据 x 向上生成 x 的高级表示 r，该高级表示 r 向下生成的 x'要尽可能与 x 一致，具体方法如下：

1）逐层构建单层神经元，这样每次都是训练一个单层网络。

2）当所有层训练完后，使用 Wake－Sleep 算法进行微调。将除最顶层的其他层间的权重变为双向的，这样最顶层仍然是一个单层神经网络，而其他层则变为图模型。向上的权重用于“认知”，向下的权重用于“生成”。然后使用 Wake－Sleep 算法调整所有的权重。让认知和生成达成一致，也就是保证生成的最顶层表示能够尽可能正确地复原底层的结点。例如，顶层的一个结点表示人脸，那么所有人脸的图像应该激活这个结点，并且这个结果向下生成的图像应该能够表现为一个大概的人脸图像。Wake－Sleep 算法分为醒（Wake）和睡（Sleep）两个部分。

① Wake 阶段，认知过程。通过外界的特征和向上的权重（认知权重）产生每一层的抽象表示（结点状态），并且使用梯度下降修改层间的下行权重（生成权重）。也就是“如果现实跟我想象的不一样，改变我的权重使得我想象的东西就是这样的”。

② Sleep 阶段，生成过程。通过顶层表示（醒时学得的概念）和向下权重，生成底层的状态，同时修改层间向上的权重。也就是“如果梦中的景象不是我脑中的相应概念，改变我的认知权重使得这种景象在我看来就是这个概念”。

由于自编码网络（Auto Encoder），广义上的自编码网络是指所有的从低级表示得到高级表示，并能从高级表示生成低级表示的近似结构，狭义上指的是其中的一种，谷歌的猫脸识别用的）有联想功能，也就是缺失部分输入也能得到正确的编码，所以上面说的算法也可以用于有监督学习，训练时，y 作为顶层网络输入的补充，应用时，顶层网络生成 y'。

2.2 DNN 应用

19 世纪 70 年代末，日本东京的 NHK 广播科学实验室开发了只有一层的神经网

络系统。它是一种视觉图像识别的神经网络，尽管它有了相当大的突破，但这种技术在学术研究领域停留了几十年。

为了认识 DNN，先看一下 DNN 的三个典型的应用。在这些应用中，读者将看到 DNN 在更广泛领域，有着巨大应用潜力。

2.2.1 提高雾天视觉能见度

毫无疑问，在有雾的情况下，诸如目标检测、跟踪等视觉活动会变得困难，导航也会变得格外困难。这是因为在雾天行驶中，大气中的雾霾微粒吸收和散射光线，使得视觉场景模糊，对比度降低，雾严重地降低了能见度。因为能见度的下降会影响驾驶员的判断，引起远距离检测系统发生错误感知，所以视觉预测和视觉增强算法有着巨大的实际价值。

一般来说，除雾算法需要同一场景没有雾的图像，或有雾图像中的突出对象，如车道标记或交通标志。汉阳大学教授 Jechang Jeong 和 Farhan Hussain 开发了除雾深度神经网络，具体算法详见图 2.2。

虽然提供的模型并没有约束训练，但事实证明，对于图像除雾有很好的效果。如图 2.3 所示，图 2.3a 是原始无雾场景，图 2.3b 是有雾场景，图 2.3c 是使用 DNN 除雾后的图片。

2.2.2 打击黑客和网络犯罪

只要使用互联网，木马、蠕虫、间谍软件和僵尸网络等恶意软件都可以使计算机陷入瘫痪，网络罪犯利用这些软件可以获取非法利益。

如果有了文件备份，可以避免经济损失，然而，恶意软件浪费了时间，造成恐慌和焦虑，因为文件的恢复需要数天时间。这样的攻击不仅影响个人的合法业务，还影响了企业，甚至政府的正常运转。尽管已经采取了许多方法进行遏制，但这些恶性活动仍然十分猖獗。

Invincea 实验室的 Joshua Saxe 和 Konstantin Berlin 使用深度神经网络识别恶意软件。他们所使用 DNN 的结构由一个输入层、两个隐藏层和一个输出层组成。输入层有 1024 个输入特征，隐藏层各有 1024 个神经元。Joshua 和 Konstantin 使用超过 400000 个二进制文件来测试他们的模型，这些数据是直接从他们的客户和内部恶意软件存储库中获得的。

他们在 0.1% 的假阳性情况下检出率为 95% 。真正了不起的是，通过直接学习

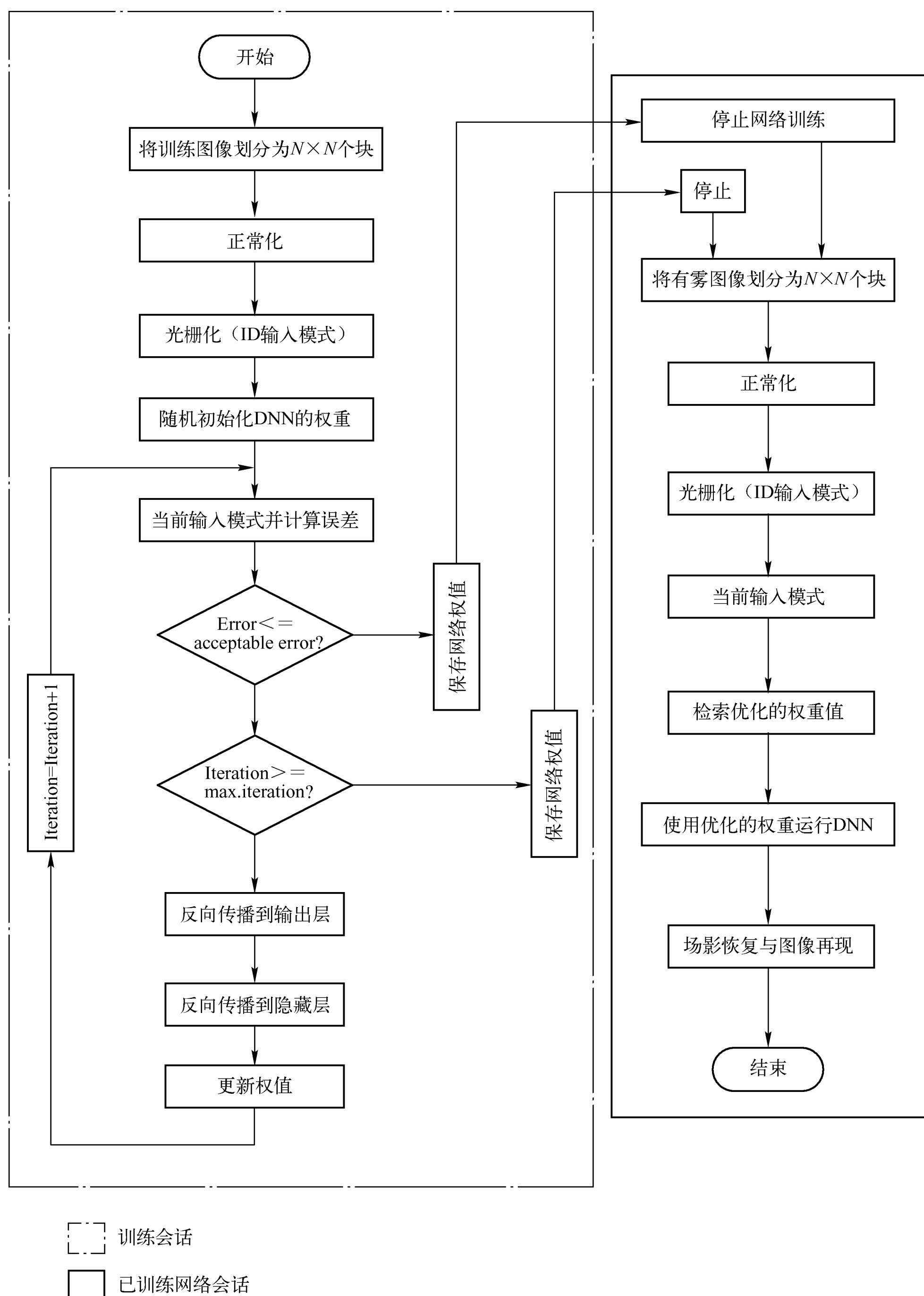


图 2.2　Hussain 的除雾 DNN 算法

所有二进制文件来达到这些结果，没有进行任何滤波、拆包或手动将二进制文件分类等操作，这是令人震惊的！

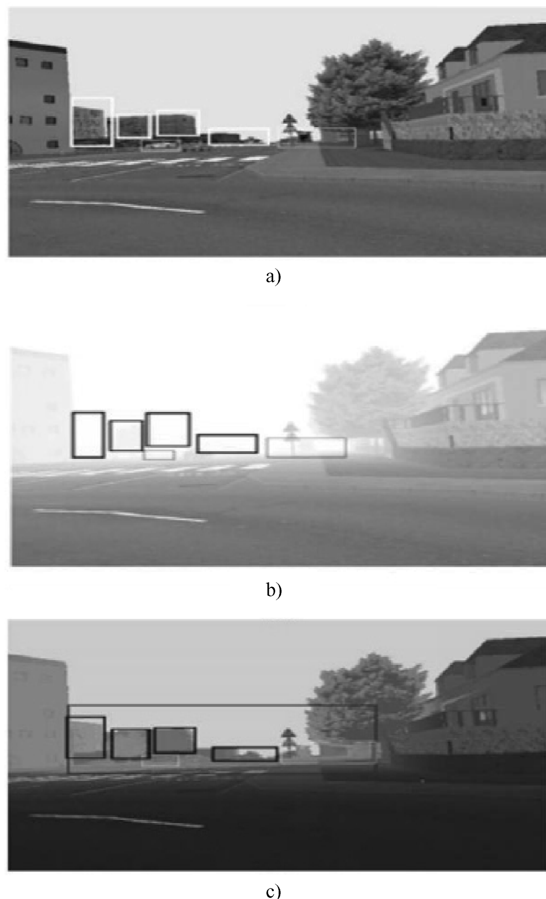


图 2.3　样本 sain 的去雾 DNN 算法

该如何解释使用深度神经网络获得的优秀结果呢？研究人员评论道：“神经网络有多种属性，适用于恶意软件检测。首先，深度神经网络允许增量学习，因此，深度神经网络不仅可以分块训练，而且可以高效地再训练，直到收集到新的训练数据；其次，通过单层的预训练，可以把标记和未标记的数据组合起来；第三，分类器非常紧凑，所以，在使用少量内存的情况下，可以快速进行预测。”

2.2.3　图像压缩

Jeong 和 Hussain 已经找出使用深度神经网络来压缩图片的方法，如图 2.4 所示。

注意，这个模型由两部分组成——“编码” 和 “解码”，对应图片压缩和解压两个阶段。第一阶段是图片的压缩；第二阶段是图片的解压缩，恢复最初的图片。输入层和输出层的神经元数量对应于图像压缩的大小，相对于原始输入，通过给最后一个隐层的神经元指定一个较小的数量，达到压缩的目的。

研究者将他们的想法应用于多个测试图片，得到了不同的图像压缩比。对于隐藏层的激活函数，使用了修正的线性单元，因为它们能“带来更好的网络归一化，并且减少了压缩－解压缩的时间”。Jeong 和 Hussain 同时也使用多层感知机激活函数来运行这个模型。

图 2.5 使用三张不同的图片展示了研究的结果。图 2.5a 是原始图像，图 2.5b 是使用修正线性单元压缩后的图片，图 2.5c 是使用多层感知机激活函数重构的图像。研究人员发现，基于 DNN 学习的压缩/解压效果很好。

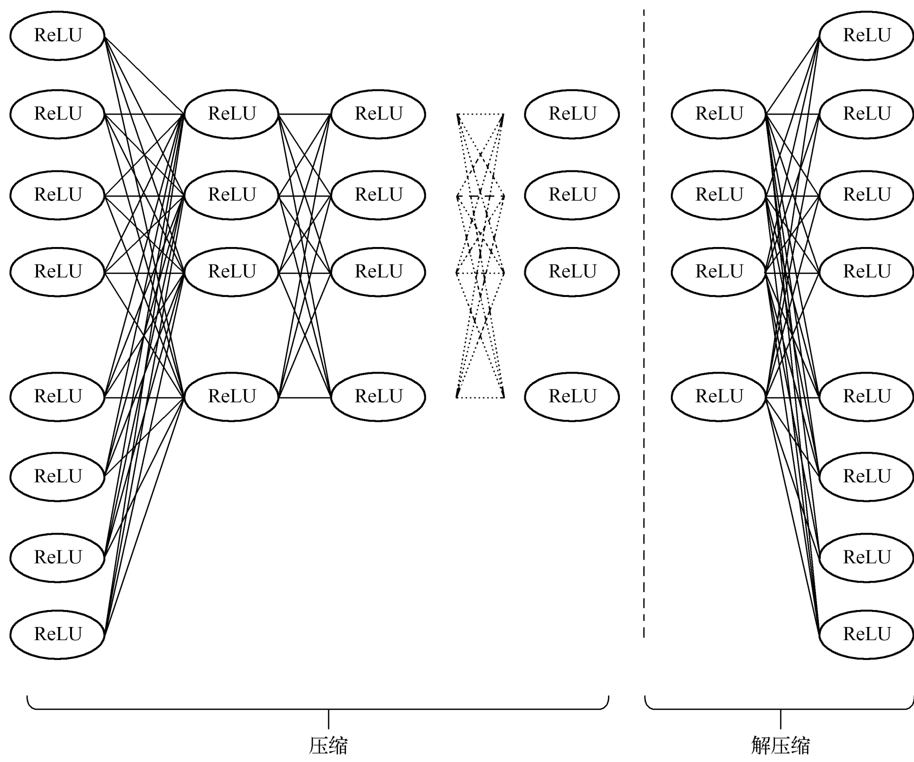


图 2.4　Jeong 和 Hussain 的图像压缩 DNN

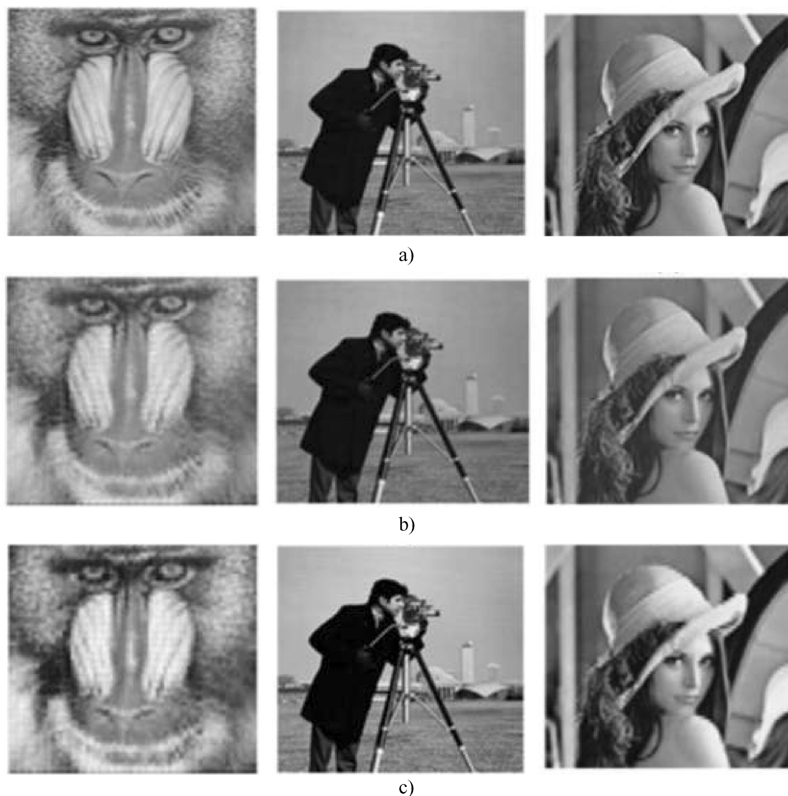
ReLU—修正的线性单元

2.2.4　函数逼近

不久前，Hornik 等人发现，一个隐藏层足以模拟任意分段连续函数。

Hornik 定理：假设 F 是 n 维坐标系中的有界连续函数。那么存在一个两层神经网络 F'，F'具有有限个隐藏层的神经元，这些神经元可以很好地表示 F。也就是，对于 F 定义域内的任意 x，都有 $|F(x)-F'(x)|<\varepsilon$。

定理意味着，对于任何连续函数 F 可以通过建立一个单隐藏层的神经网络来计



a)

b)

c)

图 2.5　基于 DNN 图像压缩结果

算。至少在理论上，对于很多问题，一个隐藏层应该足够了。

当然，在实践中还是有些问题。首先，真实世界的决策函数可能不是连续的，对于非连续函数，这个定理没有指定所需的隐藏层的神经元数量。看来，对于很多实际问题，需要多个隐藏层来进行准确分类和预测。尽管如此，这个定理仍旧有一些实用的价值。

【例 2.1】 使用 R 来创建一个 DNN 实现函数 $y = x^2$ 的近似。

（1）加载用到的包

```
> library( neuralnet)
```

（2）描述 $y = x^2$

```
> set. seed (2016)                                    #用来保证结果的重复性
> attribute <- as. data. frame( sample( seq( -2,2,length =50) ,50,
replace = FALSE) ,ncol =1)                           # attribute 为属性变量 x
> response <- attribute ^2                            #response 为响应变量 y
```

第二行产生了 50 个 −2 ~2 的观察值，其结果保存在 R 对象 attribute 中。第三行使用 R 对象 response 保存计算结果 $y=x^2$。

将 attribute 和 response 组合成数据框对象（用数据框表示数据是一个好办法，会使 R 编码变得简单）：

```
> data <- cbind(attribute,response)
> colnames (data) <- c("attribute","response")
```

查看数据的前 10 个观测值：

```
> head (data,10)
attribute          response
1  -1.2653061 1.60099958
2  -1.4285714 2.04081633
3 1.2653061 1.60099958
4  -1.5102041 2.28071637
5  -0.2857143 0.08163265
6  -1.5918367 2.53394419
7 0.2040816 0.04164931
8 1.1020408 1.21449396
9  -2.0000000 4.00000000
10  -1.8367347 3.37359434
> plot(data)  #画出散点图
```

不出所料，响应变量正是属性变量的平方。它的可视化图像如图 2.6 所示。

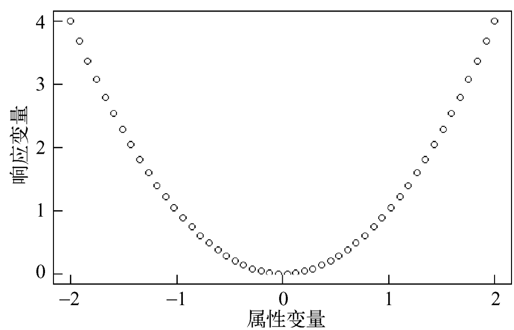


图 2.6　$y=x^2$ 模拟数据的图像

（3）建模

建立一个包含两个隐藏层、每层由三个神经元组成的 DNN，具体的方法如下：

```
> fit <- neuralnet (response ~ attribute,data = data,hidden = c(3,3),threshold =0.01)
```

公式 response ~ attribute 遵循标准的 R 实践，阈值很大程度上要根据应用的模型来定。图 2.7 显示的模型用了 3191 步使误差收敛到 0.012837。

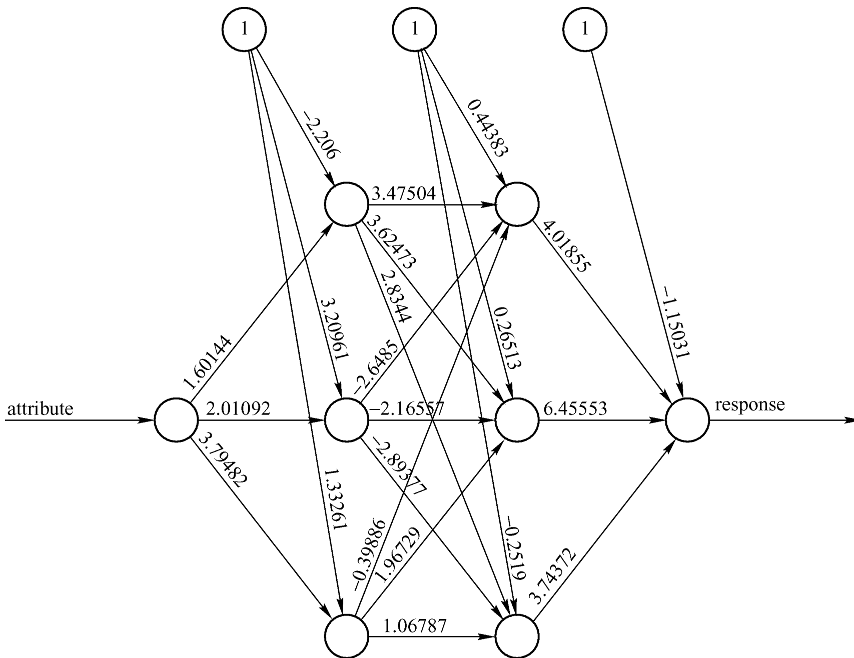


图 2.7　$y=x^2$的 DNN 模型

（4）模型部署

使用测试样本来看在进行函数近似时，模型是多么的好。在 -2 ~2 之间产生 10 个观察值，然后将结果保存在 R 对象 testdata 中。

```
> testdata <- as. matrix(sample(seq( -2,2,length = 10),10,replace = FALSE),ncol = 1)
```

使用 compute 函数完成预测：

```
> pred <- compute (fit,testdata)
```

注意：想知道在 R 对象中哪些属性是可用的，只需要输入：

```
attributes (object_name)
```

例如，想要知道属性 pred，只要输入：

```
> attributes (pred)
$names
[1] " neurons " "net. result "
```

使用$net. result 可以获取 pred 的预测值：

```
> result <- cbind(testdata,pred$net . result,testdata ^2)         #组合数据
> colnames (result) <- c("Attribute","Prediction","Actual")     #命名列名
> round(result,4)                                                 #精确到四位小数
```

	Attribute	Predication	Actual
[1,]	1. 5556	2. 4010	2. 4198
[2,]	0. 6667	0. 4183	0. 4444
[3,]	0. 2222	0. 0665	0. 0494
[4,]	-1. 1111	1. 2346	1. 2346
[5,]	2. 0000	3. 9595	4. 0000
[6,]	-1. 5556	2. 4187	2. 4198
[7,]	-0. 6667	0. 4394	0. 4444
[8,]	-0. 2222	0. 0525	0. 0494
[9,]	-2. 0000	3. 9440	4. 0000
[10,]	1. 1111	1. 2563	1. 2346

```
> plot(result)   #画出散点图
```

预测和拟合值如图 2. 8 所示，数据显示 DNN 提供了一个很好的模型，虽然不确切，但十分接近实际的函数。

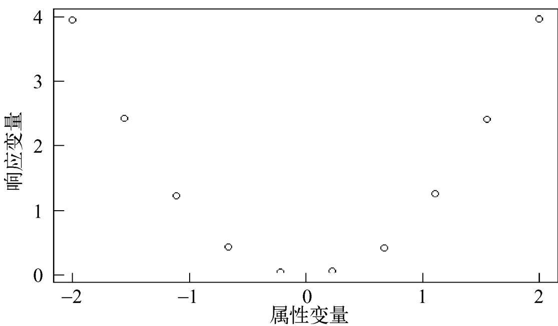


图 2. 8　DNN 预测值和拟合值

2. 3　DNN 应用需要注意的一些问题

2. 3. 1　神经元数量

DNN 的潜力取决于每层神经元数量。对于这个问题的一种解决方法是考虑从数

据包含的变量中提取的特征数。

一般的做法是每层使用更多的神经元来检测数据中更精细的结构。然而，使用的隐藏神经元越多，过拟合的风险越高，即：结果在样本中表现非常好，在非样本中却很差。

在构建 DNN 模型时，为了最好的泛化能力，DNN 应该使用尽可能少的神经元来解决问题。训练模式的数量越大，同时要保持 DNN 的泛化能力，所使用的神经元的数量也就越大。

2.3.2　最佳层数的选择

在构建和使用 DNN 时，面临的主要困难之一是选择适当的层数，该问题具有一定的挑战性。虽然在训练样本上训练 DNN 的误差非常小，但是在非训练样本可能表现得非常糟糕，讨论最新趋势的文章很少揭示这个问题的真相。

在教科书中，我们所看到的都是成功的模型，对于没有经验的人来说，觉得构建模型很容易、很简单。实际上，如果你已经花了一些时间部署商业模型，那么你一定会在模型开发阶段投入相当多的时间、精力和智力资本。

不幸的是，即使二十个模型在训练集上表现得都相当好，也只有五六个模型是真的好。因此，每一个模型都有可能在测试样本上遭遇惨败；更糟的是，应用模型可能会扼杀你的机会。因此，选择适当数量的层至关重要。

找到最佳层数的本质是一个模型选择问题。它可以使用传统的模型选择技术，最常用的方法之一就是反复试验法，其次是使用系统的全局搜索和选择标准。如果将每个隐藏层视为一个特征检测器，那么层越多，就可以学习更复杂的特征检测器。这催生了一个直截了当的经验法则，函数越复杂，使用的层就越多。

2.3.3　训练时间过长

在大数据集上运行神经网络需要耐心。由于传统神经网络算法本质上是梯度下降法，它所要优化的目标函数是非常复杂的，因此，必然会出现“锯齿形现象”，这使得算法低效；又由于优化的目标函数很复杂，它必然会在神经元输出接近 0 或 1 的情况下，出现一些平坦区，在这些区域内，权值误差改变很小，使训练过程几乎停顿；传统神经网络模型中，不能使用传统的一维搜索法求每次迭代的步长，而必须把步长的更新规则预先赋予网络，这种方法也会引起算法低效。

2.3.4 过拟合

一般情况下，训练能力差时，预测能力也差，并且一定程度上，随训练能力的提高，预测能力也提高。但这种趋势有一个极限，当达到此极限时，随训练能力的提高，预测能力反而下降，即出现所谓“过拟合”现象。此时，网络学习了过多的样本细节，而不能反映样本内含的规律。

2.4 DNN 应用技巧

（1）Dropout 能增加成功概率

随机忽略一部分隐藏层神经元的过程称为 Dropout（见图 2.9）。更为正式的陈述是：对于每个隐藏层的神经元，以概率 p 随机地从网络中省略。由于神经元是随机选择的，因此将为每个训练实例选择不同的神经元组合。

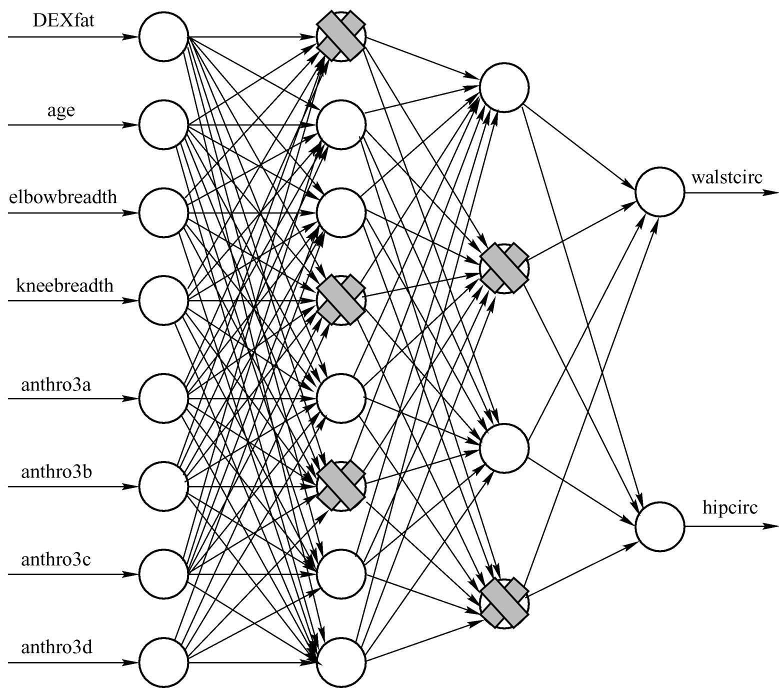


图 2.9　DNN Dropout 策略

这个想法非常简单，并在每层建立弱学习模型。弱学习模型本身具有低预测能力，然而许多弱模型的预测可以被加权并组合以产生具有更强预测能力的模型。

事实上，Dropout 非常类似于 Bagging 的机器学习技术背后的想法。Bagging 是一种模型平均方法，其中多数表决是从训练数据的 Bootstred 样本上训练的分类器中取得的。事实上，可以将 Dropout 视为多个神经网络模型的隐式 Bagging。因此，Dropout 可以被认为是在大量不同神经网络上执行模型平均有效的方式。

DNN 的能力主要来自于每个神经元，这些神经元在 DNN 中作为独立的特征检测器。然而，在应用中，要解决的另一个问题是共线性问题，即两个或多个变量高度相关。这意味着变量包含类似的信息；特别地，其中一个变量可以从其他具有非常小误差的变量线性地预测。实质上，一个或多个变量在统计上是冗余的。共线性问题可以通过从模型中 Dropout 一个或多个变量来解决。

通过在训练的前馈阶段 Dropout 神经元的激活来阻止隐藏层神经元的共线性。Dropout 也可以应用于输入层，在这种情况下，算法将随机忽略某些输入。

经验告诉我们，Dropout 并不一定会影响未来的表现，但也不能绝对地保证能提高性能，值得一试。在开发 DNN 模型时，需记住以下三点：

1）通过创建多个路径以在整个 DNN 中实现校正分类。

2）Dropout 越多，训练期间引入的噪声越多；这降低了学习速度。

3）在非常大的 DNN 模型上，Dropout 似乎能展现最大的好处。

（2）从反向传播算法中受益

正向传播计算 DNN 的所有层的每个结点的权重。反向传播是计算所有训练样本的误差，并且调整权重。

分块反向传播是加速神经网络计算的一种常用方法。它涉及同时计算多个训练样本上的梯度，而不是在原始随机梯度下降算法中发生的针对单独样本的计算。

注意，块尺度越大，运行模型所需的内存越多。为了了解分块反向传播计算效率，假设有一大小为 500 的数据，同时有 1000 个训练样本。它只需要 2 次迭代来完成 1 个结点的计算。

一个常见的问题是，什么时候应该使用分块技术？答案取决于建立的 DNN 规模。模型中的神经元越多，分块反向传播的潜在益处越大。

另一个常见的问题是块的最佳规模。在选择最佳块尺寸方面需要一定的经验。经验告诉我们，最好的建议是尝试不同的值，以了解哪些对样本和 DNN 架构有效。

(3) 早期停止的简单计划

如果能更容易提取测试样本可接受的性能，则应该提早停止 DNN 训练。这是早期停止的概念，其中样本被分成三组：训练集、验证集和测试集。

训练集用于训练 DNN，训练误差通常是单调函数，每次迭代都会减小，图 2.10 说明了这种情况，在前 100 次迭代期间，误差迅速下降，然后在接下来的 300 次迭代中下降速率更慢，随后变为恒定值。

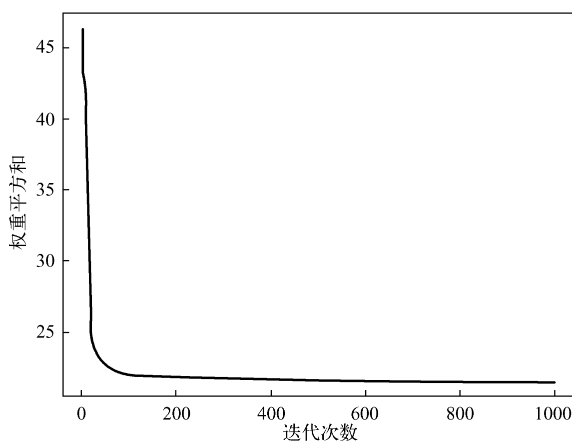


图 2.10　网络的迭代误差

验证集用于评估模型的性能。验证误差通常在早期阶段急剧下降，同时网络快速学习出函数形式，但随后误差增加，这表明模型开始过拟合。

当验证集上误差最低时停止训练，然后在测试样本上使用验证模型，对于减少过拟合是非常有效的。

2.5　单响应变量 DNN 的 R 实现

【例 2.2】利用 neuralnet 包预测波士顿郊区房价。

(1) 加载需要使用的包

```
>library (neuralnet)
>library (Metrics)
>library(dplyr)
>library(plyr)
```

数据集为 MASS 包中波士顿居民对空气质量改善付款意愿调查。

```
> data("Boston",package = "MASS")
> data  <- Boston
```

R 对象 Boston 包含 506 行和 14 列，每列对应一个特定的变量。表 2.1 给出 Boston 数据集每个变量的细节。

表 2.1　Boston 数据集

变量名	描　述
crim	城镇人均犯罪率
indus	城镇每英亩非零售商占比
nox	氮氧化物浓度
rm	平均居住的房间数量
age	1940 年之前建成的私有建筑比例
dis	到达波士顿就业中心平均距离
tax	有价财产税税率
ptration	城镇小学教师比例
lstat	低学历人口数（百分比）
medv	自有房屋数

```
> keeps  <-  c("crim","indus","nox","rm","age","dis","tax","ptratio","lstat",
"medv")                              #保留变量
> data  <- data[ keeps ]             #根据保留变量生成数据集
```

R 对象 data 现在只包含接下来要使用的变量，响应变量为 mdev。接下来，使用 apply 方法来快速查看一下保留的数据中是否有缺失值。

```
>apply(data,2,function(x) sum(is.na(x)))
crim  indus  nox  rm  age  dis  taxptratio    lstat    medv
 0      0     0    0    0    0       0          0        0     0
```

可以看出，没有缺失值!

执行代码：

```
>par(mfrow = c(3,3))
>for(i in 1:9){
>   plot(data[,i],data[,10])
>}
```

显示响应变量 medv 和输入变量之间的关系，如图 2.11 所示。

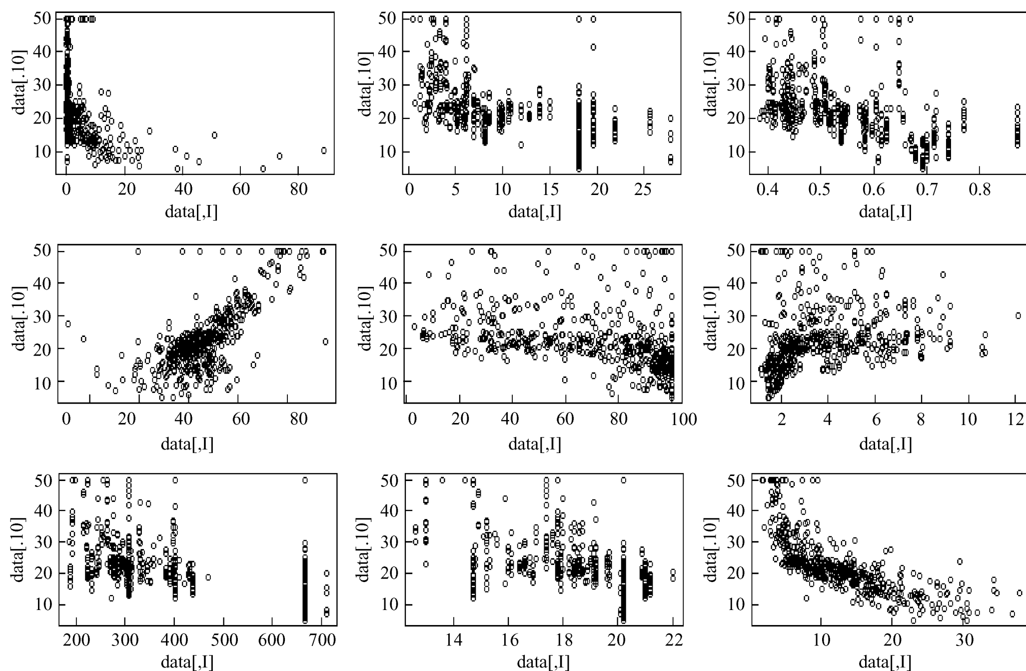


图 2. 11　medv 和输入变量的关系

使用 506 行数据样本中的 400 个作为训练样本：

```
> set. seed (2016)
> n = nrow( data)                          #统计数据行数
> train <- sample (1:n,400,FALSE)          #数据集中随机取 400 个样本作为训练集
```

注意，R 对象 train 包含了训练样本中数据行的位置（下标）。它不包含实际观察数据。

（2）建模

使用 neuralnet 包的函数构建 DNN，公式存储在名为 f 的 R 对象中。响应变量 medv 将针对剩余的九个属性“回归”。

```
> f <- medv ~  crim + indus + nox + rm + age + dis + tax + ptratio + lstat        #存储公式
```

DNN 可以使用神经网络函数进行拟合。实现如下：

```
> fit <- neuralnet (f,                  #模型公式
    data = data[ train, ] ,             #使用的数据集
    hidden = c( 10,12,20) ,             #隐藏层数和每层结点数
    algorithm = " rprop + " ,           #使用的学习算法( 具有回溯的弹性反向传播算法)
    err. fct = " sse " ,                #平方误差的总和( 误差评估策略)
```

```
act. fct = "logistic",              #激活函数
threshold = 0.1,                    #阈值
linear. output = TRUE)              #输出神经元使用的激活函数(线性)
```

注：如果想使用传统的反向传播，可以设置 algorithm = “backprop”。如果使用这个选项，还需要指定学习率（即 learningrate = 0.01）。

（3） 预测

DNN 预测可以直接使用神经网络包提供的 compute 函数，唯一需要做的事情就是将它包含的 DNN 模型与测试数据传递过去，实施方法如下：

```
> pred  <-  compute (fit,data[ - train,1:9])
```

注意， - train 是返回训练样本以外的行号。图 2.12 显示了线性回归（实线）的拟合值和预测值。虽然有一些离群值，但是该预测模型看起来相当好。这里可以通过性能标准来验证。接下来以计算平方相关系数、均方误差（mse）和均方根误差（rmse）为例进行说明：

```
> round(cor(pred $net. result,data[ - train,10])^2,6)
[1,]   0.809458         #因为平方相关系数小于 0.9,所以模型还有待提高
> mse (data [ - train,10],pred $net .  result)
[1]   0.2607601849
> rmse (data [ - train,10],pred $net .  result)
[1] 0.5106468299
```

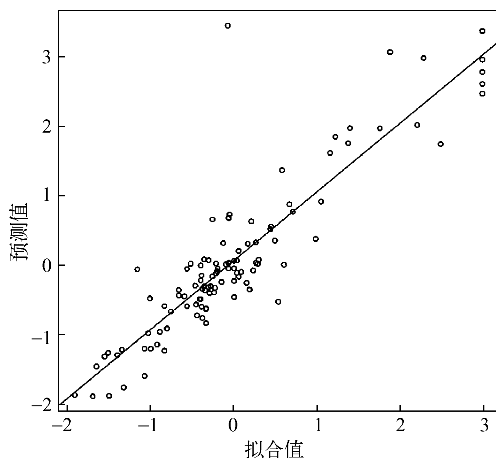


图 2.12　预测值和拟合值

2.6 多响应变量 DNN 的 R 实现

图像处理中，在单个 DNN 中通常要建立多个响应变量（像素）。然而，在数据科学领域，DNN 模型通常只包含一个响应变量。

在同一组属性上建立多个响应变量联合模型的关键统计学原理是，它可以在回归类型模型中取得更有效的参数估计。简而言之，更有效的估计会得到更快的决策，因为这时在统计推断方面将会有更大的信心。可以想到多个涵盖商业、工业和研究的应用实例，其中在同一组属性上建立多个响应变量的模型是有用的。

【例 2.3】使用 TH. data 包中的数据 bodyfat 来构建 DNN。

该数据最初由 Garcia 等人收集，以通过测量皮肤厚度、周长以及男性和女性的骨宽度来开发用于体脂预测的可靠的回归方程。最初的研究收集了 117 名健康德国受试者的数据，其中有 46 名男性和 71 名女性。体脂数据包含 71 个女性的 10 个变量的数据，见表 2.2。

表 2.2 bodyfat 数据集

变量名	描　述
DEXfat	脂肪
age	年龄
waistcirc	腰围
hipcirc	臀围
elbowbreadth	肘宽
kneebreadth	膝宽
anthro3a	统计学习理论
anthro3b	统计学习理论
anthro3c	统计学习理论
anthro4	统计学习理论

（1）加载使用的包和数据

```
> library (neuralnet)
> require (Metrics)
> data("bodyfat",package = "TH. data ")
```

（2）数据准备

由于处理的是一个相当小的数据集，只有 71 个观测值，所以使用 60 个观测值

作为训练样本。

```
> set. seed (2016)
> train <- sample (1:71,60,FALSE)        #构造训练集
```

下一步，标准化观察值，用于构建 DNN 的公式存储在 R 对象 f 中：

```
> scale_bodyfat <- as. data. frame( scale( log( bodyfat) ) )
> f <- waistcirc + hipcirc ~ DEXfat + age + elbowbreadth + kneebreadth + anthro3a + anthro3b
+ anthro3c + anthro4
```

注意使用 waistcirc + hipcirc ~ 来表示两个响应变量。通常，对于 k 个响应变量 $\{r_1,\cdots,r_k\}$，使用 $r_1+r_2+\cdots+r_k$ ~ 来表示。

（3）建模

使用两个隐藏层的模型来拟合，其中第一个隐藏层有 8 个神经元，第二个隐藏层有 4 个神经元。其余参数与已经讨论过的类似：

```
> it <- neuralnet (f,
    data = scale_bodyfat[ train, ] ,
    hidden = c(8,4) ,
    threshold =0. 1 ,
    err. fct = "sse" ,
    algorithm = "rprop + " ,
    act. fct = "logistic" ,
    linear. output = FALSE
    )
```

该模型只需要 45 步就会收敛，其平方误差的总和等于 25. 18。

然后，复制 scale_bodyfat 中的数据，并将赋值给 without_fat，这样在 scale_bodyfat 中保留一份原始值，稍后将会使用到它。注意，此时也需要使用 NULL 参数从这个新的 R 对象中删除响应变量：

```
> without_fat <- scale_bodyfat
> without_fat $waistcirc <- NULL
> without_fat $hipcirc <- NULL
```

（4）模型部署

现在把模型应用在测试样本上。第二行打印出预测值（仅显示前几个值）：

```
> pred  <-  compute (fit,without_fat [ -train,])
> pred $net. result
```

```
              [,1]                [,2]
48   0. 8324509945946   0. 7117765670627
53   0. 1464097674972   0. 0389520756586
58   0. 7197107887754   0. 6215091479349
62   0. 8886019185711   0. 8420724129305
```

至此，具有两个响应变量的 DNN 已成功构建。

（5）模型评估

为了与线性回归模型做比较，需要建立两个模型。

模型 1：

```
> fw  <-  waistcirc ~ DEXfat + age + elbowbreadth + kneebreadth + anthro3a +
anthro3b + anthro3c + anthro4
> fh  <- hipcirc ~ DEXfat + age + elbowbreadth + kneebreadth + anthro3a + anthro3b +
anthro3c + anthro4
```

现在使用训练数据运行每个模型：

```
> regw  <- linReg  <- lm(fw,data = scale_bodyfat [train,])
> regh  <- linReg  <- lm(fh,data = scale_bodyfat [train,])
```

使用测试样本预测：

```
> predw  <- predict (regw,without_fat [ -train,])
> predh  <- predict (regh,without_fat [ -train,])
```

DNN 模型与线性回归模型相比较，DNN 模型响应变量的均方差统计量可以如下计算：

```
> mse(scale_bodyfat [ -train,10],pred $net.  result [,1])
```

```
[1]   0. 5376666652
```

Galton 回归模型均方差：

```
> mse(scale_ bodyfat [ -train,10],predw)
```

```
[1]   0. 3670629158
```

Galton 的腰围的线性回归模型优于深度学习模型。

臀部尺寸模型性能怎么样呢？以下是 DNN 的均方差度量：

```
> mse(scale_bodyfat [ - train,10],pred $net . result [ ,2])
[1]   0.5530302859
```

Galton 的臀部尺寸回归模型的均方差度量：

```
> mse(scale_bodyfat [ - train,10],predh)
0.5229634116
```

此时，两者速度较为接近，但线性回归模型仍然有较小的均方误差，性能略微优于 DNN 模型 1。

模型 2：

当然，在实践中，要使用不同的学习算法、不同的神经元数量来建立多种模型；线性回归模型结果仅作为一个有用的基准。为了说明这一点，使用 deepnet 包建立传统反向传播算法 DNN。

首先，将属性变量赋值 R 对象 X 中，并将响应变量赋值 R 对象 Y 中：

```
> set. seed (2016)
> X = as. matrix (without _fat [train,])            #构造属性变量
> Y = as. matrix (scale_bodyfat [train,3:4])        #构造响应变量
```

具有 2 个隐层的 DNN 模型如下：

```
> fitB  <- nn. train (x = X,y = Y,
initW = NULL,
initB = NULL,
hidden = c(8,4),
activationfun = " sigm ",
learningrate = 0.02,
momentum = 0.74,
learningrate_scale = 1,
output = " linear ",
numepochs = 970,
batchsize = 60,
hidden_dropout = 0,
visible_ dropout = 0)
```

使用测试样本得到预测值，并存储在 R 对象 predB 中：

```
> Xtest  <-  as. matrix (without _fat[ - train,])
> predB  <-  nn. predict (fitB,Xtest)
```

腰围和臀围的均方误差计算过程如下：

```
>mse (scale_bodyfat [ -train,10],predB[ ,1])
[1]   0.1443659185
>mse(scale_bodyfat [ -train,10],predB[ ,2])
[1]   0.1409938484
```

依据均方误差，DNN 模型 2 优于线性回归模型。

注：任何一种方法的优劣都不是绝对的，应该选择一种符合问题性质的方法。

腰围和臀围作为响应变量，可视化如图 2.13 所示。属性的可视化如图 2.14 所示。

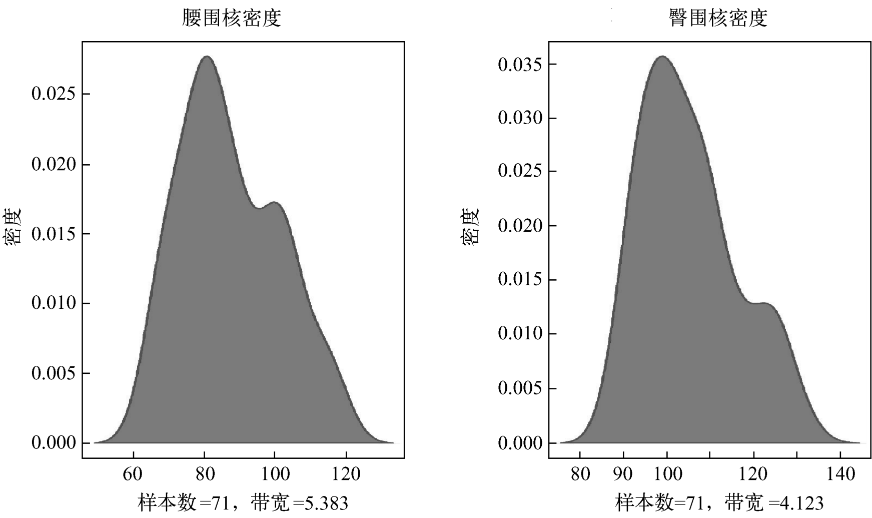


图 2.13　响应变量分布图

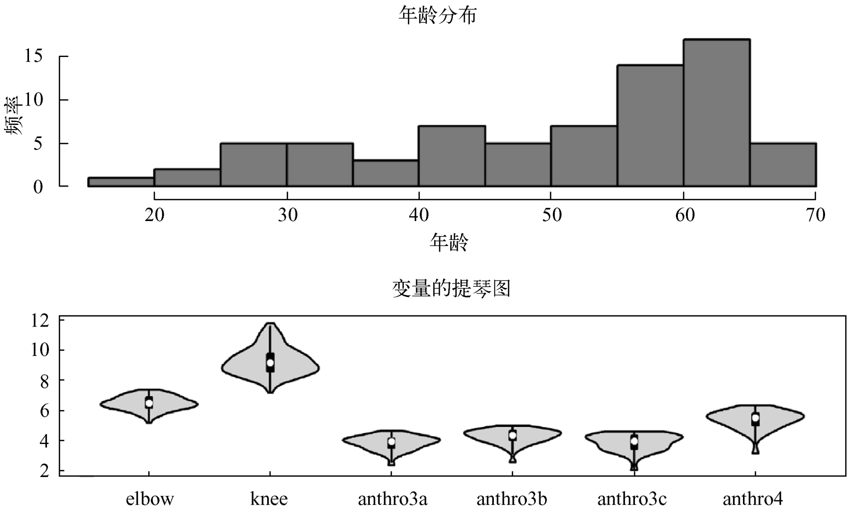


图 2.14　DNN bodyfat 属性

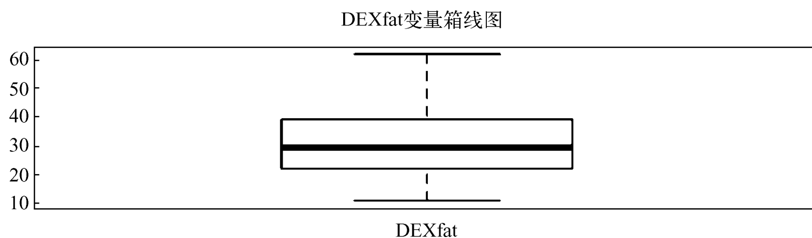


图 2. 14　DNN bodyfat 属性（续）

2.7　学习指南

值得注意的是，虽然 DNN 对特征工程的要求相对较低，但训练时间复杂度较大，且权重可解释性非常差，不易调试。因此，对于一个新的应用，比较好的方法是先用 Logistic Regression 线性模型，等迭代成熟了，再尝试 DNN 模型。

第 3 章　卷积神经网络 CNN

3.1　CNN 原理

卷积神经网络是近年发展起来，并引起广泛重视的一种高效识别方法。20 世纪 60 年代，Hubel 和 Wiesel 在研究大脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网络（Convolutional Neural Networks，CNN）。现在，CNN 已经成为众多科学领域的研究热点之一，特别是在模式识别领域，由于该网络避免了对图像的复杂前期预处理，可以直接输入原始图像，因而得到了更为广泛的应用。K. Fukushima 在 1980 年提出的新识别机是卷积神经网络的第一个实现网络。随后，更多的科研工作者对该网络进行了改进。其中，具有代表性的研究成果是 Alexander 和 Taylor 提出的“改进认知机”，该方法综合了各种改进方法的优点并避免了耗时的误差反向传播。

一般地，CNN 的基本结构包括两层：其一为特征提取层，每个神经元的输入与前一层的局部接受域相连（局部感知），并提取该局部接受域的特征；其二是特征映射层，网络的每个计算层由多个特征映射组成，每个特征映射是一个平面，平面上所有神经元的权值相等（权值共享）。特征映射结构采用 Sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数。卷积神经网络中的每一个卷积层都紧跟着一个用来求局部平均与二次特征提取的计算层，这种特有的二次特征提取结构减小了特征分辨率。

CNN 主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于 CNN 的特征检测层通过训练数据进行学习，所以在使用 CNN 时，避免了显示的特征抽取，而隐式地从训练数据中进行学习；再者，由于同一特征映射面上的神经元权值相同，

所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。

在图像处理中，往往把图像表示为像素的向量，比如一个 1000 × 1000 的图像，可以表示为一个 1 000 000 的向量。在图 1. 4 中提到的神经网络中，如果隐藏层数目与输入层一样，也是 1 000 000 个神经元，那么输入层到隐藏层的参数数据为 1 000 000 × 1 000 000 = 10^{12}，这样就太多了，基本无法训练。所以图像处理要想训练成神经网络，必先减少参数加快速度。

3. 1. 1　局部感知

卷积神经网络有两种方法可以降低参数数目，第一种方法叫作局部感知。一般认为，人对外界的认知是从局部到全局的，而图像的空间联系也是局部的像素联系较为紧密，而距离较远的像素相关性则较弱。因而，每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息。网络部分连通的思想，也是受启发于生物学里面的视觉系统结构。视觉皮层的神经元就是局部接受信息的（即这些神经元只响应某些特定区域的刺激）。如图 3. 1 所示，图 3. 1a 为全连接，图 3. 1b 为局部连接。

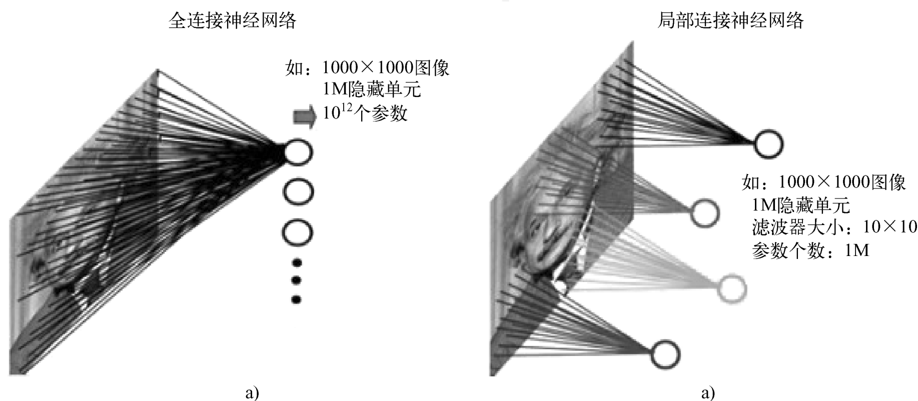


图 3. 1　CNN 结构

在图 3. 1b 中，假如每个神经元只和 10 × 10 个像素值相连，那么权值数据为 1 000 000 × 100 个参数，减少为原来的千分之一。而那 10 × 10 个像素值对应的 10 × 10

个参数，其实就相当于卷积操作。

3.1.2 权值共享

通过局部感知后参数仍然过多，那么就可以采用第二种方法，即权值共享。在上面的局部连接中，每个神经元都对应 100 个参数，一共 1 000 000 个神经元，如果这 1 000 000 个神经元的 100 个参数都是相等的，那么参数数目就变为 100 了。

怎么理解权值共享呢？可以把这 100 个参数（也就是卷积操作）看成是提取特征的方式，该方式与位置无关。这其中隐藏的原理是：图像的一部分的统计特性与其他部分是一样的。这也意味着在这一部分学习的特征也能用在另一部分上，所以对于这个图像上的所有位置，都能使用同样的学习特征。

更直观一些，当从一个大尺寸图像中随机选取一小块，比如 8 ×8 作为样本，并且从这个小块样本中学习到了一些特征，这时可以把从这个 8 ×8 样本中学习到的特征作为探测器，应用到这个图像的任意地方中去。特别是，可以用从 8 ×8 样本中所学习到的特征与原本的大尺寸图像作卷积，从而对这个大尺寸图像上的任一位置获得一个不同特征的激活值。

如图 3.2 所示，展示了一个 3 ×3 的卷积核在 5 ×5 的图像上做卷积的过程。每个卷积都是一种特征提取方式，就像一个筛子，将图像中符合条件（激活值越大越符合条件）的部分筛选出来。

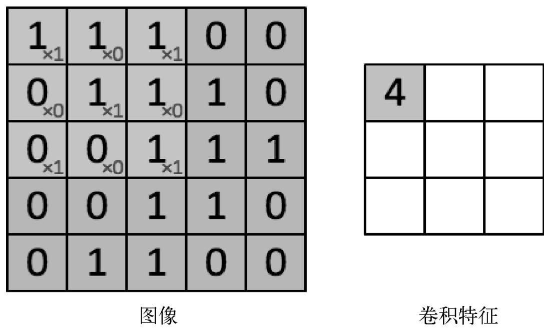


图 3.2 卷积运算过程

3.1.3 多卷积核

当只有 100 个参数时，如果只用 1 个 100 ×100 的卷积核，显然，特征提取是不

充分的，这时可以添加多个卷积核，比如 32 个卷积核，可以学习 32 种特征。有多个卷积核时的情形如图 3.3 所示。

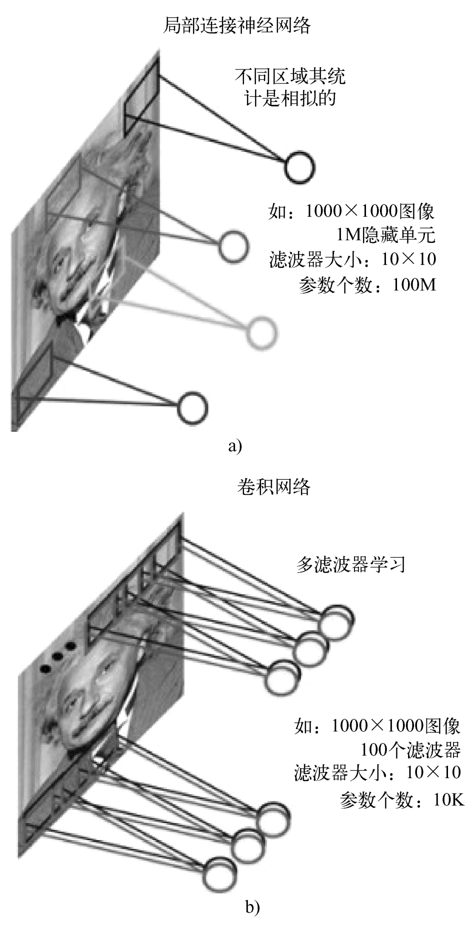


图 3.3　多卷积核示意图

图 3.3b 中不同灰度表明不同的卷积核。每个卷积核都会将图像生成为另一幅图像。比如两个卷积核就可以生成两幅图像，这两幅图像可以看作是一张图像的不同的通道，如图 3.4 所示。

图 3.4 展示了在四个通道上的卷积操作，有两个卷积核，生成两个通道。其中需要注意的是，四个通道上每个通道对应一个卷积核，先将 W^1 忽略，只看 W^0，那么在 W^0 的某位置 (i,j) 处的值，是由四个通道上 (i,j) 处的卷积结果相加然后再取激活函数值得到的。

$$h_{ij}^k = \tanh((W^k * x)_{ij} + b_k)$$

所以，在图 3.4 由 4 个通道卷积得到 2 个通道的过程中，参数的数目为 4×2×2

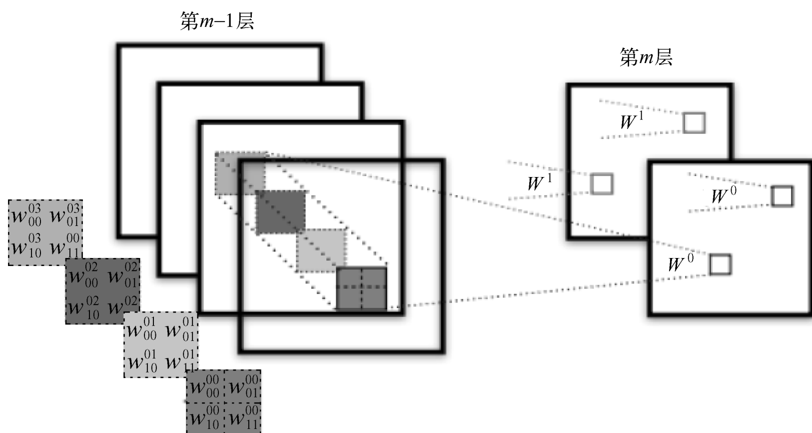


图 3.4　在四个通道上的卷积操作

×2 个，其中 4 表示 4 个通道，第一个 2 表示生成 2 个通道，最后的 2×2 表示卷积核大小。

3.1.4　池化

在通过卷积获得了特征之后，下一步希望利用这些特征去做分类。理论上讲，人们可以用所有提取得到的特征去训练分类器，例如 Softmax 分类器，但这样做面临计算量的挑战。例如，对于一个 96×96 像素的图像，假设已经学习得到了 400 个定义在 8×8 输入上的特征，每一个特征和图像卷积都会得到一个(96－8＋1)×(96－8＋1)＝7921维的卷积特征，由于有 400 个特征，所以每个样例都会得到一个 7921×400＝3 168 400 维的卷积特征向量。学习一个拥有超过 300 万特征输入的分类器十分不便，并且容易出现过拟合。

为了解决这个问题，首先回忆一下，之所以决定使用卷积后的特征是因为图像具有一种“静态性”的属性，这也就意味着在一个图像区域有用的特征极有可能在另一个区域同样适用。因此，为了描述大的图像，一个很自然的想法就是对不同位置的特征进行聚合统计，例如，人们可以计算图像一个区域上的某个特定特征的平均值（或最大值）。这些概要统计特征不仅具有低得多的维度（相比使用所有提取得到的特征），而且还会改善结果（不容易过拟合）。这种聚合的操作就叫作池化（Pooling），有时也称为平均池化或者最大池化（取决于计算池化的方法），如图 3.5 所示。

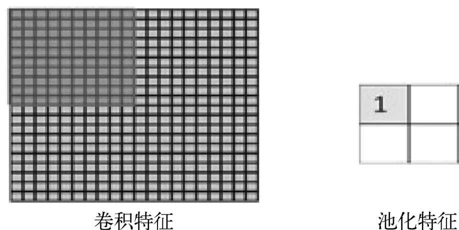


图 3.5　池化特征示意图

3.2　多层卷积

在实际应用中，往往使用多层卷积，然后再使用全连接层进行训练。使用多层卷积的原因是因为一层卷积学到的特征往往是局部的，层数越高，学到的特征就越全局化。

3.2.1　ImageNet－2010 网络结构

ImageNet LSVRC 是一个图片分类的比赛，其训练集包括 127 万多张图片，验证集有 5 万张图片，测试集有 15 万张图片。本节截取 2010 年 Alex Krizhevsky 的 CNN 结构进行说明，该结构在 2010 年取得冠军，top－5 错误率为 15.3%。值得一提的是，在 2016 年的 ImageNet LSVRC 比赛中，取得冠军的 GoogNet 已经达到了 top－5 错误率 6.67%。可见，深度学习的提升空间还很巨大。

图 3.6 即为 Alex 的 CNN 结构图。需要注意的是，该模型采用了 2 个 GPU 并行结构，即 5 个卷积层都是将模型参数分为 2 部分进行训练的。

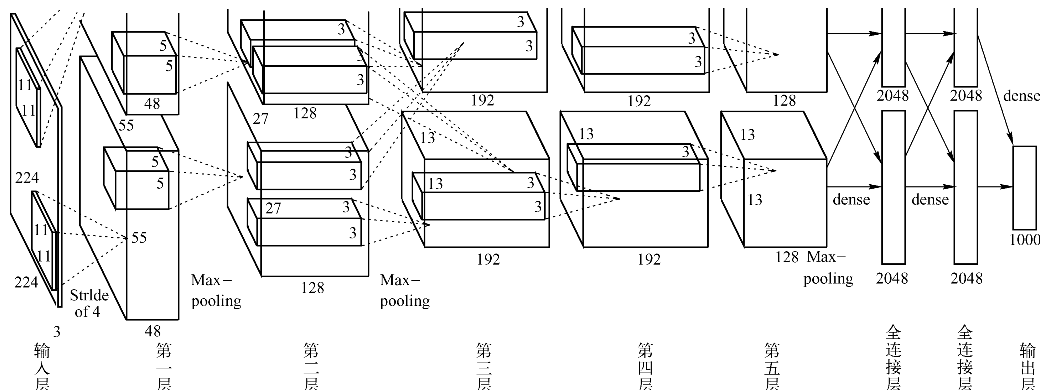


图 3.6　Alex 的 CNN 结构图

模型的基本参数如下：

输入层：224 ×224 大小的图片，3 通道。

第一层卷积：5 ×5 大小的卷积核 96 个，每个 GPU 上 48 个。

第一层 Max – pooling：2 ×2 的核。

第二层卷积：3 ×3 卷积核 256 个，每个 GPU 上 128 个。

第二层 Max – pooling：2 ×2 的核。

第三层卷积：与上一层是全连接，3 ×3 的卷积核 384 个。分到两个 GPU 上各 192 个。

第四层卷积：3 ×3 的卷积核 384 个，两个 GPU 各 192 个。该层与上一层连接没有经过 pooling 层。

第五层卷积：3 ×3 的卷积核 256 个，两个 GPU 上各 128 个。

第五层 Max – pooling：2 ×2 的核。

全连接层：4096 维，将第五层 Max – pooling 的输出连接成一个一维向量，作为该层的输入。

全连接层：4096 维。

输出层（Softmax 层）：1000 维，每一维是图片属于该类别的概率。

3. 2. 2 DeepID 网络结构

DeepID 网络结构是香港中文大学的 Sun Yi 开发出来用来学习人脸特征的卷积神经网络。每张输入的人脸被表示为 160 维的向量，学习到的向量经过其他模型进行分类，在人脸验证试验上得到了 97. 45% 的正确率，更进一步地，原作者改进了 CNN，又得到了 99. 15% 的正确率。

DeepID 网络结构与 ImageNet 网络结构的参数类似，所以，这里只解释一下 DeepID 与 ImageNet 的区别。

DeepID 网络结构最后只有一个全连接层，然后就是 Softmax 层了。在全连接层，以第四层卷积和第三层 Max – pooling 的输出作为全连接层的输入，这样可以学习到局部的和全局的特征。

3. 3 CNN 的 R 实现

从公开的材料上看，目前还没有成熟的 CNN 包，而 MATLAB 里面有很方便的

conv2，相信不久的未来会发布与 CNN 相应的 R 包，并相信 R 语言做这个也绝对不可能会比 MATLAB 弱。

从 CNN 原理看出，学习过程中需要多层迭代，时间复杂度极高，因此对 CNN 进行了优化，优化后 CNN 称为极限学习机（一种单层神经网络）。

极限学习机有以下优点：

1）由于极限学习机求取权值的时候只是计算一个广义逆，因此训练速度比基于梯度的学习算法快很多。

2）基于梯度的学习算法存在很多问题，比如学习速率难以确定、局部网络最小化等，极限学习机有效地改善了此类问题，在分类过程中取得了更好的效果。

3）与其他神经网络算法不同，极限学习机在训练过程中，选择激活函数过程中可以选择不可微函数。

4）极限学习机算法训练过程并不复杂，只需要三步就可以完成整个的学习过程。

【例 3.1】 通过极限学习机预测。

```
> library(elmNN)
> set.seed(1234)
> Var1  <- runif(50,0,100)
> sqrt.data  <-  data.frame(Var1,Sqrt = sqrt(Var1))
> model  <- elmtrain.formula(Sqrt ~ Var1,data = sqrt.data,nhid = 10,
    actfun = "sig")
> new  <-  data.frame(Sqrt = 0,Var1 = runif(50,0,100))
> p  <-  predict(model,newdata = new)
```

3.4 学习指南

“深度学习”是为了让层数较多的神经网络可以有效训练，因此演化出一系列新的结构和方法。在新的网络结构中，最著名的就是 CNN，它解决了传统较深的网络参数太多，很难训练的问题，使用了“局部感知”和“权值共享”的概念，大大减少了网络参数的数量。关键是这种结构确实很符合视觉类任务在人脑上的工作原理。

新的结构还包括 DBN、LSTM、ResNet 等。

新的方法包括：新的激活函数（ReLU），新的权重初始化方法（逐层初始化、XAVIER 等），新的损失函数，新的防止过拟合方法（Dropout、BN 等）。这些方面主要都是为了解决传统的多层神经网络的某些不足：梯度弥散、过拟合等。

第4章　递归神经网络 RNN

4.1　RNN 原理

递归神经网络（Recurrent Neural Networks，RNN）不同于 FNN（Feed - forward Neural Networks）。RNN 神经元与神经元之间相互连接，信息在这些神经元之间构成一个多向传输的循环。这种神经网络具有时间特性，可以记忆先前神经网络的状态。这让它具有能够按着时间的推移不断学习，执行分类任务，预测未来发展状态的功能。因此，RNN 可用作分类、随机序列建模和与记忆有关的任务。

RNN 的一种简单形式如图 4.1 所示。在 FNN 的隐藏层与输出层之间加入了一个延迟神经元（Delay）。延迟神经元有着存储感知的能力，因此它可以记忆前一阶段的活跃值，而在下一阶段将这些数值回馈给网络。

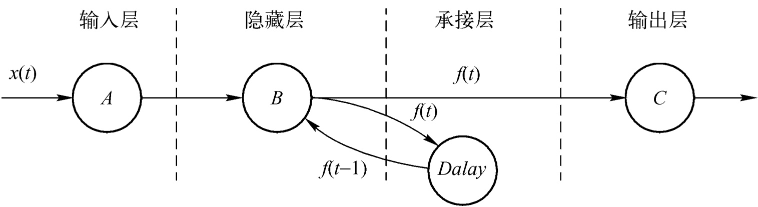


图 4.1　简单 RNN 结构

RNN 在多层传感器基础上增强一个或多个承接层（Context）之间的联系。承接层之间的神经元数量和隐藏层之间的神经元数量相等。另外，承接层的神经元和隐藏层的神经元是完全连接在一起的。

4.2　Elman 网络

Khatib 提出的 Elman 网络是为了预测每小时太阳的辐射，如图 4.2 所示。它是一

个有着 8 个输入特性（经度、纬度、温度、日照度、湿度、月、日、小时）的三层网络；五个隐藏层、五个交互神经元和两个用来预测全球太阳辐射和漫射的神经元。

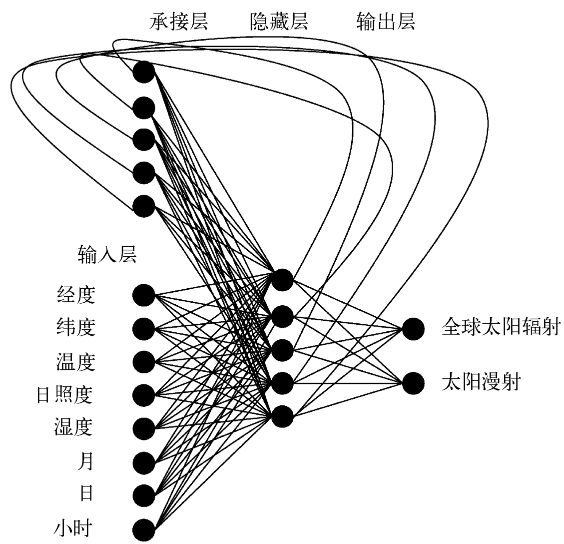


图 4.2　预测每小时太阳辐射的 Elman 网络

4.2.1　承接层神经元的作用

承接层神经元通过存储隐藏层神经元的值来记忆先前网络的内部状态。存储值将延迟一个结点时间并且在下一个结点时间作为网络额外的输出。

4.2.2　信息流动

通过图 4.3 来认识 Elman 网络模型。假设有一个由两个神经元组成的神经网络，在网络的每一层只有一个神经元，每个神经元都有一个偏置，第一个神经元的偏置为 b_1，第二个神经元的偏置为 b_2。神经元之间的连接权值重分别是 w_1 和 w_2，并且作用于函数 f_1 和 f_2。由于只有两个神经元，输出 Y 和输入 X 的函数关系式如下：

$$Y = f_2(w_2 f_1(w_1 X + b_1) + b_2)$$

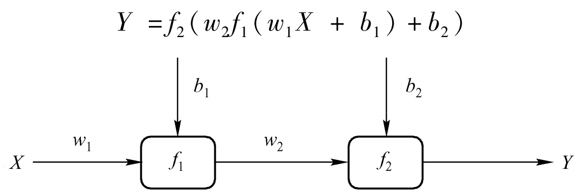


图 4.3　两个神经元的 Elman 网络图

在图 4.4 所示的 Elman 网络中，隐藏层有一个神经元接收来自某神经元（比如 x）的信号并反馈到神经元 x 的神经元 *Delay*（图 4.4 中的神经元 C）。从神经元 *Delay* 传出的信号在反馈到网络之前，会经过一个时间点的延时与 w_2 相乘。时间 t 输出的函数关系式如下：

$$Y[t]=f_2(w_3f_1(w_1X[t]+w_2Delay+b_1)+b_2)$$

其中 $Delay=Y_1[t-1]$。

在训练的过程中应当对连接权值和偏差进行迭代调整，以减少网络的错误，通常情况下应慎重使用均方误差。如果训练中的误差偏大，则应该使用另一种迭代方法。通过这个简单的例子可以看出，隐藏层和输入层是完全连接并且展现出递归的关系。

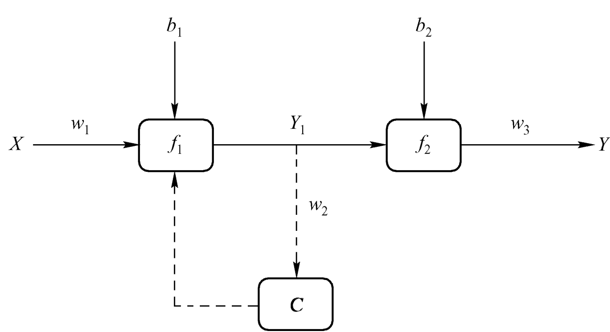


图 4.4　两结点 Elman 网络

Elman 神经网络在实际应用中非常有效，主要是用来预测给定序列的输出，具有动态特性和短时间内记忆功能。

4.2.3　Elman 网络应用

从故障预测、天气预测到股市预测，都可以看到 RNN 的应用。

（1）天气预测模型

各行各业对准确的天气预测都有浓厚的兴趣。农民依据天气状况来进行农作物的种植和收割；运输部门依据气象信息，来决定是否关闭或者打开特定的运输通道；个人关注天气情况是为了确定他们的日常活动能否正常进行。

Maqsood、Khan 和 AbRaham 共同开发的 Elman 神经网络用来预测温哥华、英国、哥伦比亚和加拿大的天气，模型尤其注重于建立用来预测每天的最高、最低气温和风速等，这几个国家一年内每天所观测到的数据都包含在其中。其中一年里面的前 11 个月的数据作为网络的训练集，最后一个月的数据作为测试集。其中，最理想的

Elman 神经网络模型是有 45 个隐藏层，激活函数为双曲正切。

在 Elman 神经网络中，气温峰值的预测和实际观测值的平均相关系数为 0.96，最低气温预测和实际观测值的平均相关系数为 0.99。并且，对于风速的预测的平均相关系数也为 0.99。

（2）逆变器故障特征识别

逆变器是城市轨道车辆的重要组成部分，如果它们发生故障会导致经济损失、班次延误，以及乘客对服务质量和可靠性的不满。基于这些原因，Yao 等应用 Elman 神经网络完成了对这个重要设备的故障检测和分类。

他构建的这个网络由 8 个输入神经元，7 个隐藏层神经元和 3 个输出层神经元组成。其中，8 个输入对应于从逆变器在频谱范围内接收的不同的故障信号。3 个输出分别对应电压波动信号、脉冲瞬态信号和频率变化信号的响应变量。研究人员说“Elman 神经网络分析技术可以用来识别城市轨道车辆逆变器的故障特征”。

（3）水质检测

水质的评估通常由水中的含氮总量（TN）、含磷总量（TP）和溶解氧（DO）的含量组成。Heyi 和 Gao 使用 Elman 神经网络预测中国第三大淡水湖——太湖的水质。他们在太湖的贡湖湾中的 3 个不同的站点测量水质，观测点的选取是随机的，观测值的 70% 组成了训练集，剩下的 30% 则组成了测试集。

检测中选择 10 个重要参数，作为水质评估的标准。他们构建了离散 Elman 网络来预测 TN、TP 和 DO。在测试中，构建了 9 个模型，而每个模型都有自己的特点。每个模型都由一个输入层、一个隐藏层和一个输出层组成。每个模型隐藏层中结点的最优个数由试验和错误率来决定。

研究人员的报告显示，TN 在站点 1、站点 2、站点 3 统计的决定系数分别为 0.91、0.72 和 0.92。TP 在站点 1、站点 2、站点 3 统计的决定系数分别为 0.68、0.45 和 0.61。尽管 TP 模型的值没有 TN 模型的值那么高，但是仍然是可以接受的。DO 在站点 1、站点 2、站点 3 统计的决定系数分别低于 0.3、0.39 和 0.83。对于这些较低的值，研究人员建议“不仅可以通过增加 3 个站点训练集和测试集的数据，还可以通过导入和湖水流动方向相关的变量来提高模型的准确性”。

（4）金融指数预测

预测金融指数（例如股市）的能力是 Elman 神经网络吸引人的地方。研究人员为四种指数分别创建了一个模型，其中，上海证券交易所（SSE）综合指数模型隐藏层有 9 个结点，台湾证券交易所（TWSE）市值加权指数模型隐藏层有 12 个结点，

韩国股票价格指数（KOSPI）和日经225指数（Nikkei225）模型隐藏层每个都有10个结点。研究人员的报道称，这四个模型和实际观察值之间的相关系数都达到了0.99，建立模型使用的数据覆盖了2000个交易日的收盘价数据。

4.3 Jordan 网络

4.3.1 Jordan 网络结构

Jordan 网络与 Elman 神经网络类似，唯一不同的是，承接层神经元的输入来自输出层的输出而不是隐藏层的输出，如图4.5所示。

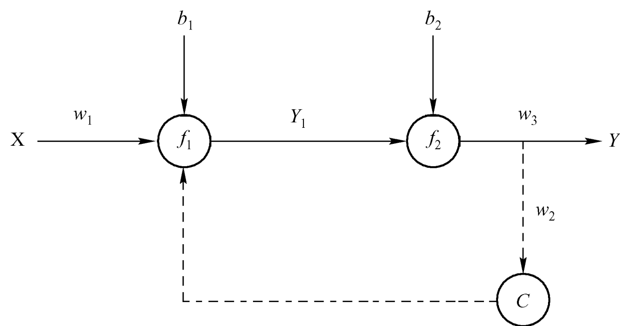


图4.5 简单的Jordan网络

激活的输出层结点不断地反馈给承接层结点，为网络提供了记忆以前状态的机制。

4.3.2 Jordan 网络应用

时间序列数据使用Jordan网络建立分类模型特别有效。下面列举一些现实世界的例子说明Jordan网络预测分析工具的使用。

（1）风速预测

准确预测沿海地区风速，对于一个大的工业区以及交通和社会海洋活动是非常重要的。例如，操纵风力涡轮机，飞机、轮船的导航。风速预测是确定预期的有用功率输出的依据。Anurag 和 Deo 构建的 Jordan 网络是为了预测印度沿海地区每日、每周和每月的风速。

从印度气象部门获得的数据覆盖了12年内沿海孟买地区以及印度西海岸。使用三个Jordan网络分别预测每日、每周和每月的风速。

这三个模型的均方误差都小于 10% 。然而，每日预测比每周的预测更准确；每周的预测比每月的预测更准确。工程师也比较了网络预测和自动回归集成移动平均（ARIMA）时间序列预测模型；他们发现“Jordan 网络预测也比传统的时间序列分析方法更准确”。

（2）蛋白质间相互作用分类

蛋白质相互作用是指生物细胞内蛋白质之间的相互作用发生的生化事件。据说，这种交互对理解疾病发病机理和开发新的治疗方法非常重要。有许多不同的方法用来研究蛋白质的相互作用，如从生物化学、量子化学、分子动力学和信号传导等，所有这些信息使得创建大型蛋白质相互作用数据库成为可能。

计算机科学家 Dilpreet 和 Singh 应用 Jordan 网络来分类蛋白质间相互作用。在他们的分析中，使用的样本来自三个现有的数据库，数据库包含了 753 个正例模式和 656 个反例模式。使用氨基酸组成的蛋白质作为 Jordan 网络的输入，分类有相互作用的蛋白质和无相互作用的蛋白质，研究表明分类精度达到 97. 25% 。

（3）西班牙语数字语音识别

神经网络已成功地解决了英语语音识别的难题。Tellez Paola 使用 Jordan 网络研究了西班牙语数字 0 ~9 的分类。

数据集为 10 个演讲中被记录的数字语音，包括 3 个女人和 3 个男人的声音。每个人被要求重复每个数字 4 遍。然后用 9 个随机初始化的 Jordan 网络训练，实现对数字分类，结果平均分类精度达到 96. 1% 。

4. 4　RNN 的 R 实现

在英国，天气总是一个聊谈的话题，因为，英格兰的天气总是在变化。你可以在一天里体验四个季节。

【例 4. 1】使用 Jordan 网络建立诺丁汉市气温预测模型。

与 Elman 网络类似，Jordan 网络模型也是对时间序列数据的建模工具。

（1）加载使用的包和数据

```
> library(RSNNS)
> library(quantmod)
> library(datasets)
> data(nottem)
```

数据是 datasets 包的 nottem 数据集，nottem 数据集包含了诺丁汉市，每年每月平均测量空气温度。首先了解一下 nottem 的数据结构：

```
> nottem
```

	Jan	Feb	Mar	Apr	May	Jun	jul	Aug	Sep	oct	Nov	Dec
1920	40.6	40.8	44.4	46.7	54.1	58.5	57.7	56.4	54.3	50.5	42.9	39.8
1921	44.2	39.8	45.1	47.0	54.1	58.7	66.3	59.9	57.0	54.2	39.7	42.8
1922	37.5	38.7	39.5	42.1	55.7	57.8	56.8	54.3	54.3	47.1	41.8	41.7
1923	41.8	40.1	42.9	45.8	49.2	52.7	64.2	59.6	54.4	49.2	36.3	37.6
1924	39.3	37.5	38.3	45.5	53.2	57.7	60.8	58.2	56.4	49.8	44.4	43.6
1925	40.0	40.5	40.8	45.1	53.8	59.4	63.5	61.0	53.0	50.0	38.1	36.3
1926	39.2	43.4	43.4	48.9	50.6	56.8	62.5	62.0	57.5	46.7	41.6	39.8
1927	39.4	38.5	45.3	47.1	51.7	55.0	60.4	60.5	54.7	50.3	42.3	35.2
1928	40.8	41.1	42.8	47.3	50.9	56.4	62.2	60.5	55.4	50.2	43.0	37.3
1929	34.8	31.3	41.0	43.9	53.1	56.9	62.5	60.3	59.8	49.2	42.9	41.9
1930	41.6	37.1	41.2	46.9	51.2	60.4	60.1	61.6	57.0	50.9	43.0	38.8
1931	37.1	38.4	38.4	46.5	53.5	58.4	60.6	58.2	53.8	46.6	45.5	40.6
1932	42.4	38.4	40.3	44.6	50.9	57.0	62.1	63.5	56.3	47.3	43.6	41.8
1933	36.2	39.3	44.5	48.7	54.2	60.8	65.5	64.9	60.1	50.2	42.1	35.8
1934	39.4	38.2	40.4	46.9	53.4	59.6	66.5	60.4	59.2	51.2	42.8	45.8
1935	40.0	42.6	43.5	47.1	50.0	60.5	64.6	64.0	56.8	48.6	44.2	36.4
1936	37.3	35.0	44.0	43.9	52.7	58.6	60.0	61.1	58.1	49.6	41.6	41.3
1937	40.8	41.0	38.4	47.4	54.1	58.6	61.4	61.8	56.3	50.9	41.4	37.1
1938	42.1	41.2	47.3	46.6	52.4	59.0	59.6	60.4	57.0	50.7	47.8	39.2
1939	39.4	40.9	42.4	47.8	52.4	58.0	60.7	61.8	58.2	46.7	46.6	37.8

没有出现任何缺失值，检查一下 nottem 数据类型：

```
> class(nottem)
[1] "ts"
```

Nottem 是一个时间序列 ts 对象，了解数据的类型是非常重要的，特别是使用日期类型或 R 的不同类型的混合。

图 4.6 显示了 nottem 观察数据的时间序列散点图。数据覆盖 1920～1939 年，虽然没有任何明显趋势，然而它确实表现出强烈的季节性。

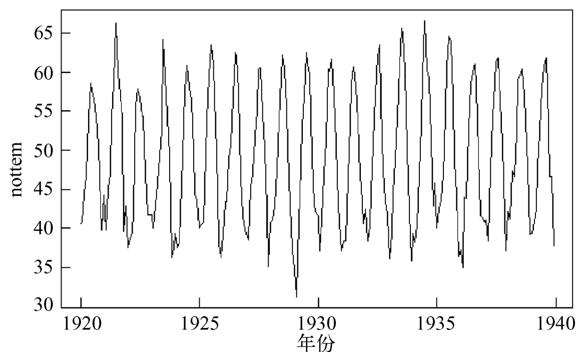


图 4.6　1920～1939 年诺丁汉市月平均温度

画时间序列散点图命令如下：

```
> plot( nottem)
```

（2）数据探索

在使用神经网络模型之前，需要归一化数据的属性。下面给出常用的四种归一化方法：

$$z_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

$$z_i = \frac{x_i - \bar{x}}{\sigma_x}$$

$$z_i = \frac{x_i}{\sqrt{SS_i}}$$

$$z_i = \frac{x_i}{x_{\max} + 1}$$

式中，SS_i是 x_i的平方和；$\bar{x}$ 是 x_i的均值；σ_x 是 x_i的标准差。因为没有任何说明性变量，所以，首先使用 log 变换数据，再使用 scale 函数标准化数据。

```
> y <- as. ts( nottem )
> y <- log( y)
> y <-  as. ts( scale( y) )
```

因为建模数据有很强的季节性特征，似乎依赖于月，所以使用滞后 12 个月的数据作为 Jordan 网络 12 个属性的输入。Quantmod 包的 Lag 函数需要的数据类型为 zoo 类。

```
> y <- as. zoo( y)
> x1 <- Lag( y,k =1)
> x2 <- Lag( y,k =2)
```

```
> x3 <- Lag(y,k = 3)
> x4 <- Lag(y,k = 4)
> x5 <- Lag(y,k = 5)
> x6 <- Lag(y,k = 6)
> x7 <- Lag(y,k = 7)
> x8 <- Lag(y,k = 8)
> x9 <- Lag(y,k = 9)
> x10 <- Lag(y,k = 10)
> x11 <- Lag(y,k = 11)
> x12 <- Lag(y,k = 12)
```

与 Elman 神经网络类似，需要删除 NA 值，最后将清洗后的数据存入 temp 中。

```
> temp <- cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12 )
> temp <- cbind(y,temp )
> temp <-  temp [ -(1:12),]
```

数据探索最后一步是使用 plot 函数可视化所有属性和响应变量的分布，如图 4.7 所示。

```
> plot(temp)
```

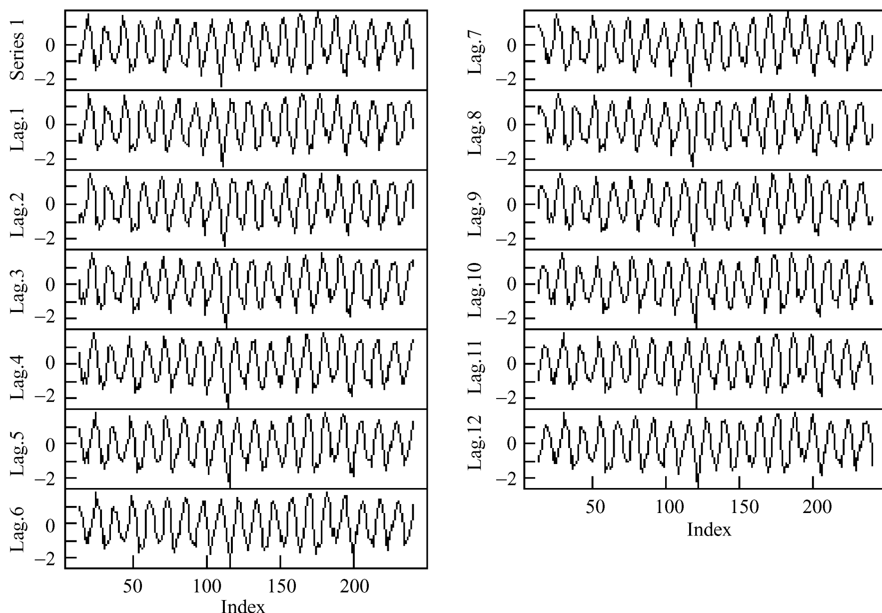


图 4.7　Jordan 网络响应变量和属性分布图

（3）训练样本选择

```
>n = nrow( temp )                              #观测数据个数
>n
[1]228
>set. seed(465)                                #设置种子
>n_train <- 190                                #设置训练集大小
>train <- sample(1:n,n_train,FALSE )           #随机选取 190 个训练样本
```

（4）建模

```
>inputs <- temp [ ,2:13]                       #属性变量
>outputs <- temp [ ,1]                         #响应变量
>fit <- jordan(inputs[train],outputs[train], #建模
      size =2,                                 #隐藏层数
      learnFuncParams = c(0. 01),              #学习率
      maxit =1000)                             #最大迭代次数
```

（5）模型部署

使用 plotIterativeError 函数显示迭代误差，如图 4. 8 所示。

```
>plotIterativeError(fit)
```

前 100 次迭代误差下降相当快，大约迭代 300 次趋于稳定。

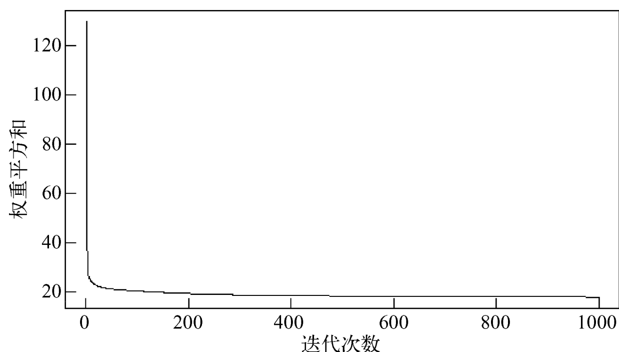


图 4. 8　训练误差与迭代次数

使用 predict 函数对测试样本进行预测，并计算测试样本响应变量和预测值之间的平方相关系数。

```
> pred <- predict(fit,inputs [ - train ])
> cor( outputs [ - train ],pred )^2
```

[1,]0. 9050079

因为平方相关系数大于 0.9，所以模型是成功的。

【例 4.2】建立一个 RNN 模型，预测在英国死于支气管炎、肺气肿、哮喘的总人数。

（1）检查机器上安装了哪些包

```
> pack <- as. data. frame( installed. packages( )[ ,c(1,3:4) ])
> rownames( pack) <- NULL
> pack <- pack[ is. na( pack$Priority ) ,1:2,drop = FALSE ]
> print( pack,row. names = FALSE )
```

（2）加载需要的包和数据

```
> library( RSNNS)
> library( quantmod)
> library( datasets)
> data( UKLungDeaths)
```

使用的数据在 datasets 包里面，所需要的数据结构为 UKLungDeaths。dataests 包是默认加载的包，所以不需要专门加载它。但是，调用它也是一种有益的练习。

UKLungDeaths 数据库包含了 3 个 1974 ~ 1979 年英国每个月支气管炎、肺气肿、哮喘的死亡人数的时间序列。第一个时间序列是死亡的总人数（ldeaths）；第二个时间序列是男性死亡人数（mdeaths）；第三个时间序列是女性死亡人数（fmdeaths）。可以通过使用 plot 函数可视化这些数据（见图 4.9），方法如下：

```
> par( mfrow = c(1,3))    #窗口划分为 1 行 3 列
> plot( ldeaths,xlab = " Year " ,ylab = " Both sexes " ,main = " Total " )
> plot( mdeaths,xlab = " Year " ,ylab = " Males " ,main = " Males " )
> plot( fdeaths,xlab = " Year " ,ylab = " Females " ,main = " Females " )
```

总而言之，随着时间的推移，可以看到死亡人数有着巨大的变化，每月死亡人数的波动清晰可见。可以观察到，1976 年出现每月最高和最低的死亡总数。有趣的是，对于男性来说，每个周期的最低点呈现下降的趋势。而进入 1979 年以后所有的

趋势都呈相当明显的下降。

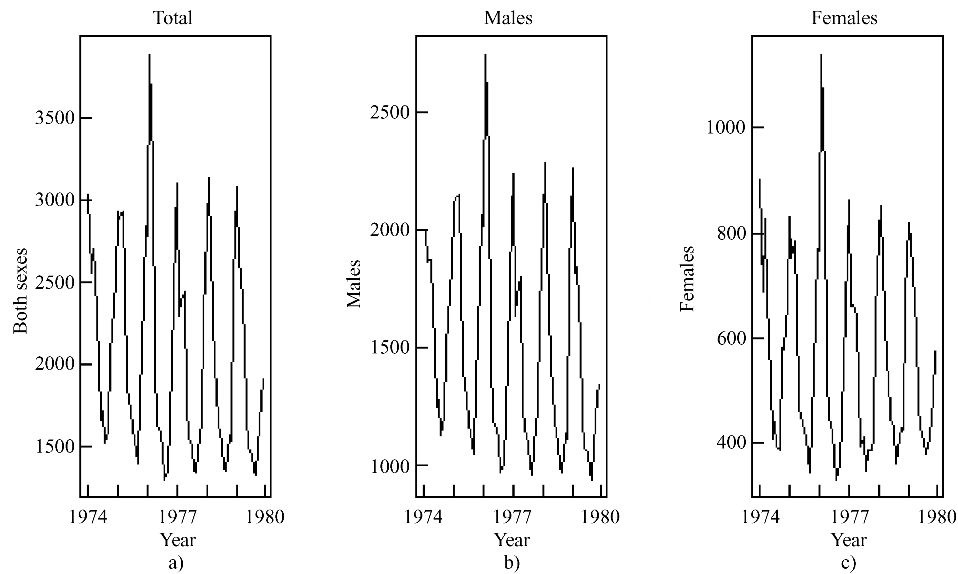


图 4.9　英国每个月支气管炎、肺气肿、哮喘死亡人数

a）总人数　b）男性死亡人数；c）女性死亡人数

（3）数据探索

因为对死亡总数的建模感兴趣，所以重点分析 ldeaths 数据结构。在此之前，快速检查缺失数据。

```
> sum( is. na( ldeaths ) )
[1]0
```

表示数据集 ldeaths 没有缺失数据，接下来，检查 ldeaths 类型。

```
> class( ldeaths )
[1]"ts"
```

表示 ldeaths 是一个时间序列对象，图 4.10 显示了时间序列图、核密度图和箱线图。

```
> par( mfrow = c(3,1))                #窗口划分为 3 行 1 列
> plot( ldeaths )
> x <- density( ldeaths )
> plot( x,main = "UK total deaths from lung diseases " )
> polygon( x,col = "green ",border = " black " )
> boxplot( ldeaths,col = " cyan ",ylab = " NumbeR of deaths per month " )
```

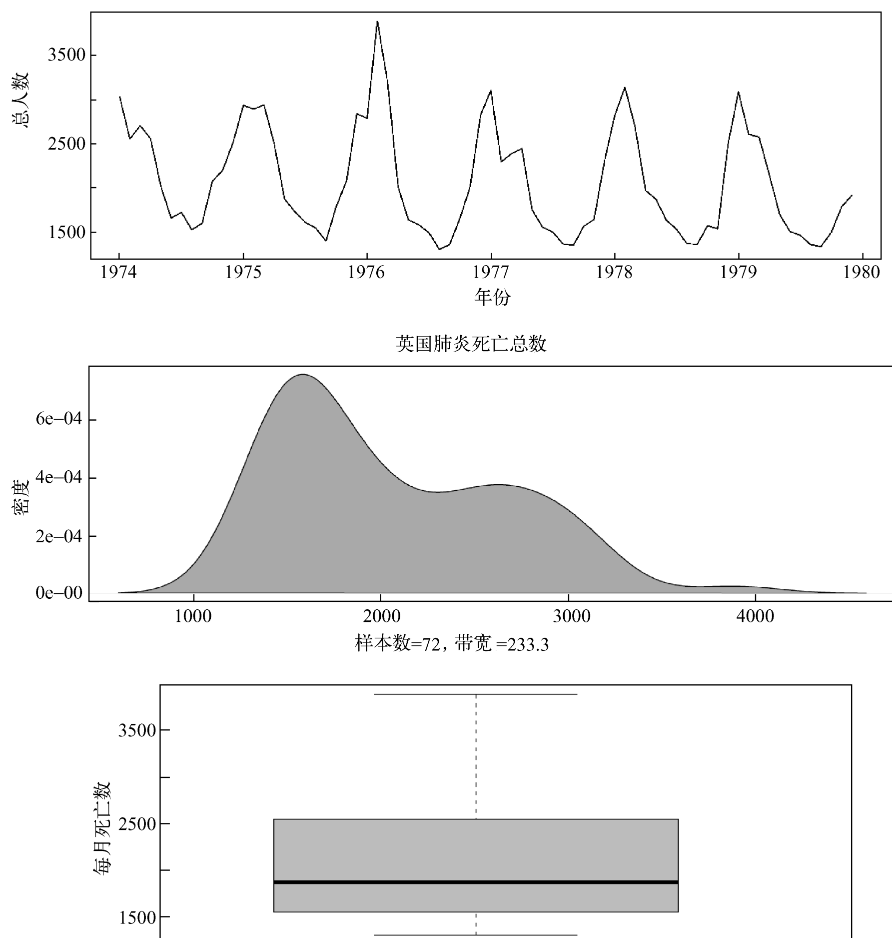


图 4.10　死亡总数的形象化总结

（4）数据转换

由于数据似乎存在某种规律，所以将其转换成一种适合进行 Elman 神经网络工作的形式。首先，将数据复制一份，存储在变量 y 中。

```
>y <- as.ts( ldeaths )
```

当使用时间序列时，一般将数据转换成对数。将数据转换成对数，数据会更加规范化。

```
>y <- log(y)
```

通过 scale 函数使数据看起来更加规范。

```
>y <- as.ts(scale(y))
```

注意，as. ts 函数确保变换后仍然是 ts 对象。

既然无法将这个数据归于其他的类别，那么就使用纯时间序列的方法。这种建模方法的主要问题是需要多少滞后变量（lags）？既然是对一年内的每个月都进行了观察，所以使用 12 作为滞后变量。实现这个操作的方法是使用 quantmod 包里面的 Lag 操作函数，这就需要将 y 转变成一个 zoo 类的对象。

```
> y <- as.zoo(y)
  x1 <- Lag(y,k = 1)
  x2 <- Lag(y,k = 2)
  x3 <- Lag(y,k = 3)
  x4 <- Lag(y,k = 4)
  x5 <- Lag(y,k = 5)
  x6 <- Lag(y,k = 6)
  x7 <- Lag(y,k = 7)
  x8 <- Lag(y,k = 8)
  x9 <- Lag(y,k = 9)
  x10 <- Lag(y,k = 10)
  x11 < - Lag(y,k = 11)
  x12 <- Lag(y,k = 12)
```

通过这个操作得到了 12 个属性作为神经网络的输入。下一步是将这 12 个观测值组合成一个数据框。

```
> deaths <- cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12 )
> deaths <- cbind(y,deaths )
```

观察 deaths 里面包含什么。

```
> head(round(deaths,2),10)
```

	Series 1	Lag. 1	Lag. 2	Lag. 3	Lag. 4	Lag. 5	Lag. 6	Lag. 7	Lag. 8	Lag. 9	Lag. 10	Lag. 11	Lag. 12
1	1. 51	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
2	0. 90	1. 51	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
3	1. 10	0. 90	1. 51	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
4	0. 90	1. 10	0. 90	1. 51	NA	NA	NA	NA	NA	NA	NA	NA	NA
5	0. 07	0. 90	1. 10	0. 90	1. 51	NA	NA	NA	NA	NA	NA	NA	NA
6	-0. 62	0. 07	0. 90	1. 10	0. 90	1. 51	NA	NA	NA	NA	NA	NA	NA
7	-0. 48	-0. 62	0. 07	0. 90	1. 10	0. 90	1. 51	NA	NA	NA	NA	NA	NA

8	-0.91	-0.48	-0.62	0.07	0.90	1.10	0.90	1.51	NA	NA	NA	NA	NA
9	-0.75	-0.91	-0.48	-0.62	0.07	0.90	1.10	0.90	1.51	NA	NA	NA	NA
10	0.17	-0.75	-0.91	-0.48	-0.62	0.07	0.90	1.10	0.90	1.51	NA	NA	NA

注意到 NA 的滞后值从 1 增大到 12。使用延迟的目的就是为了得到这个结果。然而，需要从数据集里面删去 NA 观测值。

```
> deaths <- deaths [ -(1:12), ]
```

现在已经准备好开始创建训练集和测试集。首先，计算出数据的行数，并且使用 set. seed 函数来输出它。

```
> n = nrow(deaths)
> n
[1]60
> set. seed(465)
```

作为一种快速的检测方法，可以看到有 60 行的观测数据可以用来作为分析。为了解决滞后值的问题，删除了 12 行的观测值，而结果也达到了预期。

使用其中的 45 行数据来建模，剩下的 15 行用来作为测试集。用如下代码随机选择一些不重复的数据：

```
> n_train <- 45
> train <- sample(1:n,n_train,FALSE )
```

（5）建模

为了使过程变得更加简单一点，可以将包含有滞后值的协方差的属性放入到 R 的 inputs 对象里面，将相应变量放到 ouputs 对象里面。

```
> inputs <- deaths[ ,2:13]
> outputs <- deaths[ ,1]
```

在每个神经网络中，配置有两个隐藏层，每个隐藏层里面都包含一个结点。并将学习速率设置为 0. 1，最大的迭代次数设置为 1000。

```
> fit <- elman(inputs [ train ],outputs [ train ],size = c(1,1),
leaRnFuncPaRams = c(0. 1),maxit = 1000)
```

若给定一个较小的数据，模型的收敛会变得很快，可以画出如图 4. 11 所示的误差函数。

```
> plotIterativeError(fit)
```

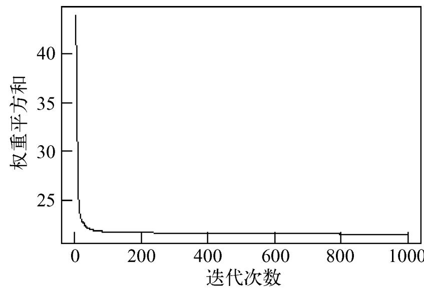


图 4. 11　Elman 函数误差图像

从图 4. 11 可以看出，误差迅速减小，大约在 500 次迭代后趋于稳定，可以使用 summaRy 函数来得到神经网络细节方面的信息。

```
> summary( fit )
```

通过这个函数可以得到很多 R 的输出值，找到输出中如下所示的部分。

unit definition section：

no.	typeName	unitName	act	bias	st	position	act func	out func	sites
1		inp1	-0. 98260	0. 15181	i	1，1，0	Act_Identity		
2		inp2	-1. 37812	-0. 32468	i	1，2，0	Act_Identity		
3		inp3	-1. 32325	0. 09987	i	1，3，0	Act_Identity		
4		inp4	-1. 05630	0. 63419	i	1，4，0	Act_Identity		
5		inp5	-0. 95449	0. 02857	i	1，5，0	Act_Identity		
6		inp6	-0. 53901	-0. 05401	i	1，6，0	Act_Identity		
7		inp7	0. 28829	-0. 04926	i	1，7，0	Act_Identity		
8		inp8	0. 93013	0. 77409	i	1，8，0	Act_Identity		
9		inp9	0. 97351	0. 66627	i	1，9，0	Act_Identity		
10		inp10	1. 56596	0. 02014	i	1，10，0	Act_Identity		
11		inp11	0. 81645	-0. 52729	i	1，11，0	Act_Identity		
12		inp12	-0. 88288	-0. 15885	i	1，12，0	Act_Identity		
13		hid11	0. 19579	-0. 76258	h	7，1，0			
14		hid21	0. 12854	-0. 71341	h	13，1，0			
15		out1	-0. 38231	2931. 21216	o	19，1，0	Act_Identity		
16		con11	0. 23859	0. 50000	sh	4，14，0	Act_Identity		
17		con21	0. 25917	0. 50000	sh	10，14，0	Act_Identity		

从第一列中，可以得到结点或者神经元的数量，而且可以知道这个网络一共有 17 行。它的第三列描述了神经元的类型，从中可以得知有 12 个输入神经元、2 个隐藏神经元和 1 个输出神经元，在承接层有 2 个神经元。第四列和第五列给出了激活函数的值和每个神经元的偏置。例如，第一个神经元的激活函数值为 -0.98260，偏置为 0.15181。

（6）模型部署

现在，在测试集上用 prRedict 函数来测试一下这个模型。

```
>pred <- predict(fit,inputs [ -  train ])
>plot( outputs[ -train],pred)
```

（7）模型评估

预测值和实际值的散点图如图 4.12 所示。

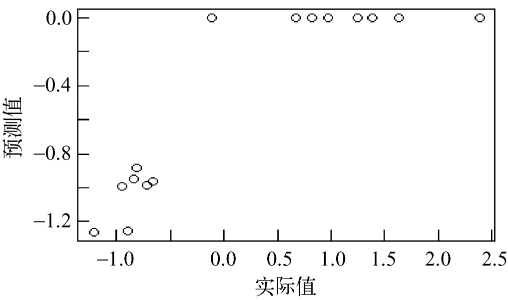


图 4.12　RNN 的实际值和预测值

```
>cor( outputs [ -  train ],pred )^2
```

[,1]

[1,]0.7845

由此可知，平方相关系数为 0.78。可以尝试不同的模型来提高这个系数，以达到理想值。试一试，通过重新建模来达到整体性能提高的效果。

4.5　学习指南

RNN 对时间序列数据的建模尤其有效。

训练 Elman 神经网络可以用 train( ) 或 adapt( )。两个函数不同之处在于，train( ) 函数应用反向传播训练函数进行权值修正，通常选用 traingdx 训练函数；adapt( ) 函

数应用学习规则函数进行权值修正，通常选用 learngdm 函数。

Elman 神经网络的可靠性要比一些其他类型网络差一些，这是因为在训练和调整时，应用误差梯度的估计值。恰恰因为这一点，构建网络时，为了达到这一精度，Elman 神经网络隐藏层神经元的数目比其他网络结构相对较多。

第 5 章　自编码网络 AE

5.1　无监督学习过程

深度学习最简单的一种方法是利用人工神经网络（Artificial Neural Network, ANN）的特点，人工神经网络本身就是具有层次结构的系统，如果给定一个神经网络，假设其输出与输入是相同的，然后训练调整其参数，得到每一层中的权重。自然地，得到了输入 I 的几种不同表示（每一层代表一种表示），这些表示就是特征。自动编码（Auto Encoder，AE）网络就是一种尽可能复现输入信号的神经网络。为了实现这种复现，AE 必须捕捉可以代表输入数据的最重要的因素，就像主成分分析（Principal Component Analysis，PCA）那样，找到可以代表原信息的主要成分。

在本章之前的神经网络中（见图 5.1a），输入的样本是有标签的，网络根据当前输出和实际值之间的差来改变前面各层的参数，直到收敛。但现在只有无标签数据，也就是图 5.1b。那么这个误差怎么得到呢？

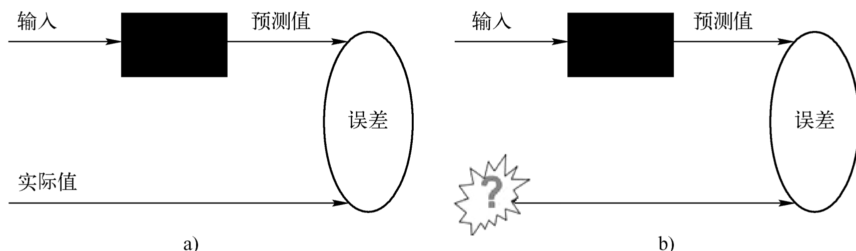


图 5.1　两种网络对比

如图 5.2 所示，将信号 input 输入到编码器（encoder），就会得到一个特征，这个特征就是输入的一个表示，那么怎么知道这个特征表示的就是输入呢？可以再加一个解码器，这时候解码器会输出一个信息，如果输出的信息和一开始的输入信号是很像的（理想情况下就是一样的），就可以相信这个特征是正确的。所以，可以通

过调整编码器和解码器的参数，使得重构误差最小，得到输入信号的第一个表示。因为是无标签数据，所以误差的来源就是直接重构后与原输入相比得到的（见图5.3）。

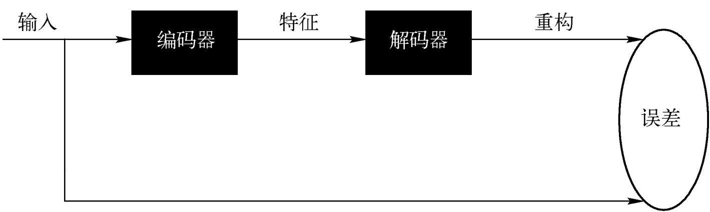


图 5.2　无监督特征学习过程

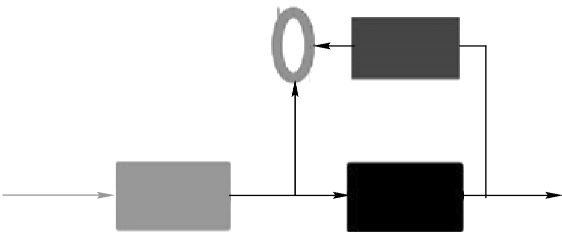


图 5.3　重构过程

5.2　AE 基本结构

自编码网络（AE）是一种无监督学习三层前向反馈神经网络，架构如图 5.4 所示，与多层感知器非常相似，它包括一个输入层、隐藏层和输出层。输入层的神经元数目和输出层的神经元数目是相同的。隐藏层神经元的数目少于或者多于输入神经元的数目。它不同于 MLP，因为输出层有着和输入层一样多的结点，而不是通过给出的 x 来训练预测一些目标值 y。

AE 由编码器和译码器组成。输入层到隐藏层的映射被称为编码器；隐藏层到输出层的映射被称为解码器。编码器采用输入属性的向量，通常通过在隐藏层的 sigmoid 激活函数将它们转化为新的特征，然后解码器将这些特征转换回原来的输入属性。举个例子，在图 5.4 中，AE 使用三个隐藏单元，每一个单元包含 sigmoid 激活函数 h_1、h_2、h_3，将属性 x_1、x_2、x_3、x_4、x_5、x_6 编码，解码后得到 $\hat{x}_i(i=1,\cdots,6)$ 的原始属性的估计。

AE 尝试学习一个 $h_{w,b}(\boldsymbol{x})\approx\boldsymbol{x}$ 的函数，也就是说，使得输出 $\hat{\boldsymbol{x}}$ 接近于输入 $\boldsymbol{x}$。为了使这个函数有意义，需要加入一些限制条件（比如限制隐藏层神经元的数目，即

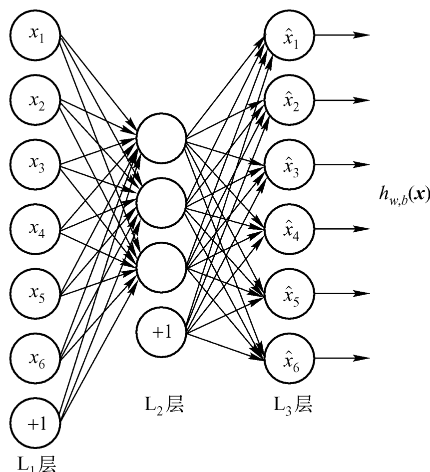


图 5.4　基本 AE 结构

稀疏性)。AE 可以学习到数据的一些压缩（降维）表示，即输入数据的另一种相关性表示。

稀疏性限制是指如果当神经元的输出接近于 1 的时候认为它被激活，而输出接近于 0 的时候认为它被抑制，那么使得神经元大部分的时间都是被抑制的限制则被称作稀疏性限制。这里假设神经元的激活函数是 sigmoid 函数。

5.2.1　降维问题

通过隐藏层学习 $h_{w,b}(\boldsymbol{x})$，可以重建原始输入 $\boldsymbol{x}$，如果隐藏层神经元的数量大于输入层神经元数量，那么隐藏层的输入映射到一个更高的维度。相似地，如果隐藏神经元的数量少于输入神经元的数量，那么 AE 的隐藏层用如此方法，本质上是压缩输入属性，使重构更有效。

压缩指原始输入属性可以用更低维度的属性表示。例如，当 AE 有 400 个输入神经元和 60 个隐藏结点，原始的 400 维输入会被近似地“重建”为来自隐藏层的 60 维输出。如果使用隐藏层的输出作为一个网络的输入代表，那么 AE 扮演了一个特征提取的作用。

事实证明，AE 可以实现多种降维技术。一个线性 AE 可以学习数据的特征向量相当于主成分分析运用到输入上。非线性 AE 能够发现更复杂的主成分。在涉及笔迹和人脸识别的降维任务中，非线性 AE 优于主成分分析。

因此，AE 可以用来学习有着最小重建损失的数据的压缩（或扩大）。正如深度学习学者杰弗里·辛顿（Hinton）所指出的“自 20 世纪 80 年代以来，已经显而易

见了，通过深度 AE 的反向传播对于非线性降维非常有效，如果能提供尽可能快的计算机、足够大的数据集和初始权值，将会足够接近一个好的解决方案。现在三个条件都满足了。不像非参数方法，AE 在数据和编码空间之间给出了双向映射，随着时间和空间的变化，训练样本都被线性地训练和微调，所以，它们可以应用到非常大的数据集。”

最著名的降维技术是主成分分析（Principal Component Analysis，PCA）。PCA 旨在利用降维的思想，把多指标转化为少数几个综合指标（即主成分），其中每个主成分都能够反映原始变量的大部分信息，且所含信息互不重复。这种方法在引进多方面变量的同时将复杂因素归结为几个主成分，使问题简单化，同时得到的数据更加科学和有效。在实际问题研究中，为了全面、系统地分析问题，必须考虑众多影响因素。这些涉及的因素一般称为指标，在多元统计分析中也称为变量。因为每个变量都在不同程度上反映了所研究问题的某些信息，并且指标之间彼此有一定的相关性，因而所得的统计数据反映的信息在一定程度上有重叠。主要方法有特征值分解、SVD、NMF 等。

主成分分析法是一种数学变换的方法，它把给定的一组相关变量通过线性变换转换成另一组不相关的变量，这些新的变量按照方差依次递减的顺序排列。在数学变换中保持变量的总方差不变，使第一变量具有最大的方差，称为第一主成分；第二变量的方差次大，并且和第一变量不相关，称为第二主成分。依次类推，I 个变量就有 I 个主成分。

如果输入属性不包含结构，那么降维是徒劳的。例如，如果输入的属性是完全随机的，那么降维是不可行的。有效的降维需要属性是相关的或者在某些程度上是有联系的。换句话说，需要一些可以用来降维数据的结构。如果结构不存在，那么使用 AE 来降维很可能会失败。

5.2.2 特征抽取

AE 是为了训练一个至少有一个隐藏层的网络来重建其输入。输出值被设置与输入值相同，即$\hat{\boldsymbol{x}}=\boldsymbol{x}$，为了学习这个恒等函数 $h_{w,b}(\boldsymbol{x})\approx\boldsymbol{x}$。AE 通过输入属性到隐藏层结点的映射来使用一个编码功能：

$$h_{w,b}(\boldsymbol{x})=f(\boldsymbol{W}\boldsymbol{x}+b\boldsymbol{h})$$

其中，$\boldsymbol{x}$ 是属性的输入向量；f 表示 sigmoid 函数，$b\boldsymbol{h}$ 是隐藏层神经元的偏置向量，$\boldsymbol{W}$ 是隐藏层权值矩阵。数据通过使用一个线性解码被重构：

$$g_{w,b}(\hat{\boldsymbol{x}}) = \lambda f(\boldsymbol{Wx} + b\boldsymbol{h}) + bg$$

在学习了权值矩阵 $\boldsymbol{W}$ 之后，每一个隐藏层神经元都代表了输入数据的某一特征。因此，隐藏层 $h_{w,b}(\boldsymbol{x})$ 可以作为一个新的特征代表输入数据。隐藏层的输出表示 $h(\boldsymbol{x})_{w,b}$ 通过解码函数 $g_{w,b}(\hat{\boldsymbol{x}})$ 重构输入 $\boldsymbol{x}$ 的近似值 $\hat{\boldsymbol{x}}$。

编码器和解码器最常用的函数是非线性 sigmoid 函数。通过梯度下降算法使用反向传播来最小化平方重建误差 $(g_{w,b}(\hat{\boldsymbol{x}}) - \boldsymbol{x})^2$，从而完成训练。

总的来说，AE 是一个试图复制其输入的人工神经网络，即目标输出就是输入。另一种稍微正式的说法是，AE 是一种前馈神经网络，试图通过在训练中设定输出等于输入来学习一个恒等函数。

5.3 稀疏自动编码网络 SAE

当然，还可以继续加上一些约束条件得到新的深度学习方法，如：假设在 AE 的基础上加上 L1 的限制（L1 主要是约束每一层中的结点大部分都为 0，只有少数不为 0，这就是 Sparse 名字的来源），就可以得到稀疏自动编码（Sparse Auto Encoder，SAE）网络。

如图 5.5 所示，其实就是限制每次得到的表达特征尽量稀疏。因为稀疏的表达往往比其他的表达要有效（人脑好像也是这样的，某个输入只是刺激某些神经元，其他大部分的神经元是受到抑制的）。

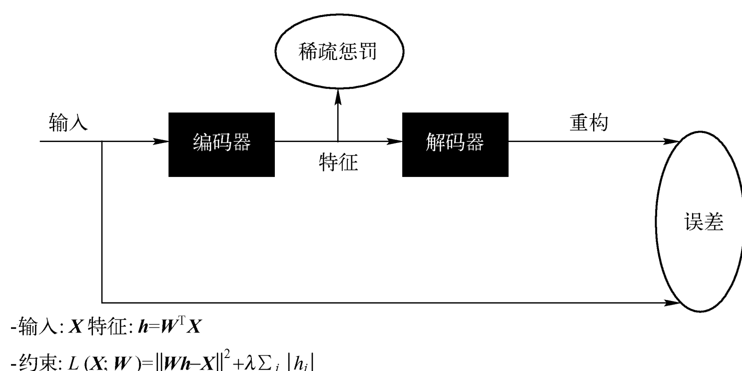


图 5.5　稀疏自动编码网络结构

通过设置隐藏神经元的数量远远大于输入神经元的数量，建立输入向量 $\boldsymbol{x}$ 的一个非线性映射，然后对它们实施一个稀疏约束。因此，训练涉及使用系数约束来学习数据的稀疏表示。最受欢迎的稀疏约束是 Kullback－Leibler（KL）散度。

5.3.1 Kullback－Leibler 散度

Kullback－Leibler 散度是测量从一个“真正”的概率分布到一个“目标”的伯努利随机变量概率分布的距离，设“真正”的概率分布为 p，伯努利随机变量为 $\dot{p}_j$，则

$$KL(p \parallel \dot{p}_j) = p\log\left(\frac{p}{\dot{p}_j}\right) + (1-p)\log\left(\frac{1-p}{1-\dot{p}_j}\right)$$

注意：当 $p = \dot{q}_j$时，KL（$p \parallel \dot{q}_j$）$=0$，否则 KL 是正值。

参数 p 是稀疏参数，通常被设置为一个小的值。它是隐藏结点的激活频率；例如，如果 $p = 0.07$，神经元 j 的平均激活是7%。使用输入属性 x_i和 $a_j^{(2)}$ 定义隐藏层神经元 j 的激活值，即

$$\dot{q}_j = \frac{1}{n}\sum_{i=1}^{n}[a_j^{(2)}(x_i)]$$

在所有训练样本中，参数$\dot{q}_j$ 是隐藏神经元 j 的平均阈值激活参数。为了计算$\dot{q}_j$，整个训练集需要向前传播来计算所有单元的激活值。其次是随机梯度下降法，使用反向传播。相对于标准的 AE，这使得它计算费时。

对于单元 j 的稀疏优化目标是 $p = \dot{q}_j$，而单元 j 是最小化重建误差平方或损失函数时通过添加下面的稀疏约束来获得的：

$$\alpha \sum_j KL(p \parallel \dot{p})$$

在 AE 损失函数中，α 是一个超函数，决定了稀疏项的相对重要性。一般来说，AE 优化目标是使权重最小化，即

$$\arg\min_{W,b} J(W,b) = L(\hat{\boldsymbol{x}},\boldsymbol{x})$$

式中，L 是一个损失函数，如平方误差或交叉熵；W 是权重；b 是偏置。SAE 优化目标是

$$J_{\text{Sparse}}(W,b) = J(W,b) + \alpha \sum_j KL(p \parallel \dot{p})$$

可以看到 SAE 优化目标多了一个稀疏约束项。

5.3.2 使用 SAE 注意事项

1）在实际中使用 SAE 网络开发的应用比使用标准的 AE 开发的应用多。主要是因为 SAE 网络在训练期间可以学习更多种类的模型，根据激活函数，规范化隐藏单元的数量和性质。

2）使用 SAE 网络进行分类的另一个理由是稀疏表示的特征能更好地为原始输入数据编码使其有识别能力。

3）如果隐藏层神经元个数比输入层神经元个数多，标准 AE 学习的是恒等函数，从而没有从输入属性中提取有用的特征。

5.4 SAE 的 R 实现

【例 5.1】建立一个 SAE 模型，压缩 R－LOG 图像并提取隐含特征。

（1）加载依赖的包和数据

```
> require(autoencoder)
> require(ripa)
```

autoencoder 包包含需要的函数来建立稀疏自编码网络。Ripa 包包含一个 R－LOG 图像，加载图像如图 5.6 所示。

```
> data(logo)
```

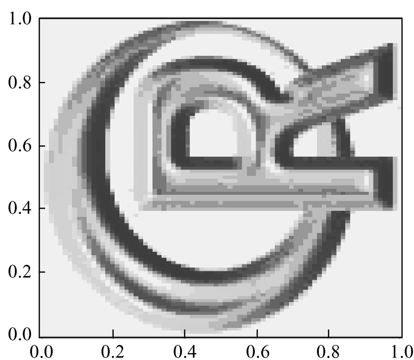


图 5.6　图像 R－LOG

查看图像的特征：

```
> logo
  size : 77 × 101
  type : grey
```

这是一个灰度大小 77 ×101 像素的图像。

（2）建模

首先，复制这张图像并赋值给 x_train。用 t( )转置图像来使得它适合 autoencoder

包的使用。

```
> x_train <- t(logo)
```

x_train 是 101 行（样本），77 列（属性）的灰度图像。

现在使用 autoencoder 函数建立 SAE 模型。

```
> set. seed(2016)
> fit <- autoencode(X. train = x_train,X. test = NULL,
    nl = 3,N. hidden = 60,
    unit. type = "logistic",
    lambda = 1e -5,
    beta = 1e -5,
    rho = 0. 3,
    epsilon = 0. 1,
    max. iterations = 100,
    optim. method = c("BFGS"),
    rel. tol = 0. 01,
    rescale. flag = TRUE,
    rescaling. offset = 0. 001)
```

第二行表示模型将保存在 R 对象 fit；并将 x_train 里的图像数据传输到函数中。参数 n1 代表层数被设置为 3。使用逻辑激活函数，隐藏结点数目为 60。lambda 是一个权重衰减参数，通常设置为一个较小的值；beta 有着相同的值，它是稀疏性惩罚项的权重。稀疏度设置为 0. 3（rho）并按正态分布 N(0，epsilon2）采样。iterations 的最大值设置为 100。注意：rescale. flag = true 统一重新调节训练矩阵 x_train，因此它的值位于 0 ~ 1 之间（logistic 激活函数）。

查看 fit 相关的属性：

```
attributes(fit)
$names
[1] "W"                               "b"
[3] "unit. type "                     "rescaling "
[5] "nl"                              "sl"
[7] "N. input "                       "N.  hidden "
[9] "mean. error. training. set"      "mean. error. test. set"
```

```
$class
[1]"autoencoder "
```

下面代码查看训练集均值误差：

```
> fit$mean. error . training . set
[1]0. 3489713
```

（3）模型预测

正如看到的，SAE 经常对提取隐藏结点的特征是有用的。使用 hidden. output = TRUE 实现预测。

```
> features <- predict(fit,X. input = x_train,hidden. output = TRUE)
```

由于隐藏结点的数目设置为 60，属性的数目为 77，特征是原始图像的紧凑表示，可视化表示为图 5. 7 所示。

```
> image(t(features$X. output))
```

转置函数 t( )用于重新定位特征来匹配图 5. 7。

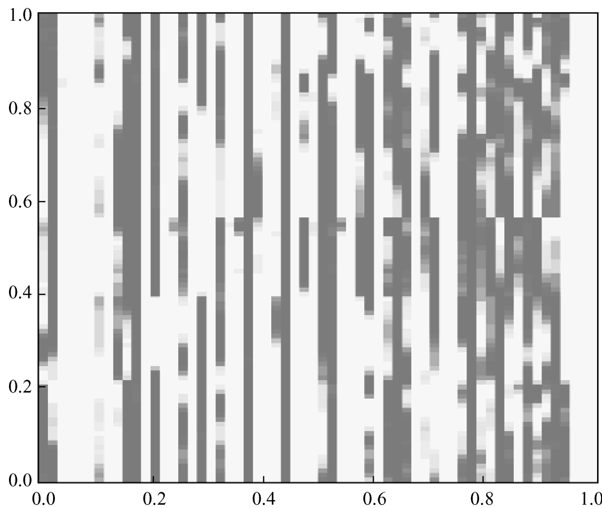


图 5. 7　从 fit 提取隐藏结点的特征

注意，使用 Nelder - Mead、准线性牛顿（Quasi - Newton）法和共轭梯度算法的 Autoencoder 函数在数据包中被称为优化函数。目前的优化方法包括：

1）“BFGS”是一种拟牛顿方法，它使用函数值和梯度来建立一个图像表面进行优化。

2）“CG”是一个共轭梯度算法，通常比 BFGS 方法更脆弱。它的主要优点是运

行时不需要存储大量的矩阵。

3）“L－BFGS－B”允许每个变量被给一个较低或者较高的限制。

压缩特征如何捕获原始图像？使用 hidden. out＝FALSE 的预测函数来重建值。

```
> pred <- predict(fit,X. input = x_train,hidden. output = FALSE )
```

均方误差显得相当小。

```
> pred$mean. error
[1]0. 3503714
```

重建图像（见图 5. 8），表明用 SAE 表示原始图像相当得好！

```
> recon <- pred$X. output
> image( t( recon ) )
```

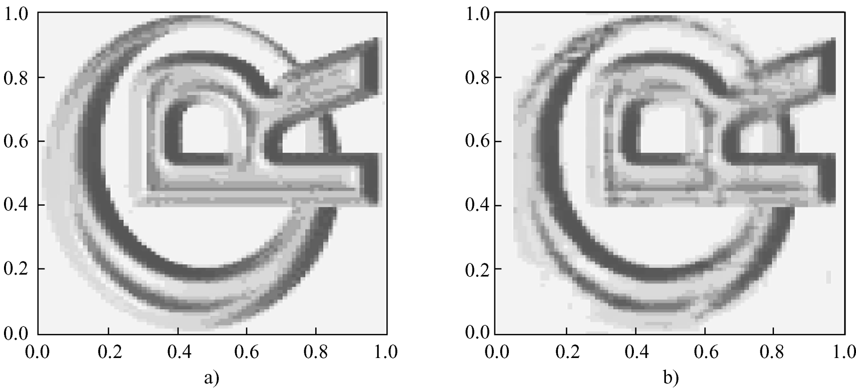


图 5. 8　原始标志和稀疏 Autoencoder 重建标志

a）原始标志　b）稀疏 Autoencoder 重建标志

【例 5. 2】用 SAE 和 R 执行一个可使用的软体动物分析。

通过本例的学习，思考如何把使用的方法来适应自己的研究。

鲍鱼、海洋蜗牛、蛤、扇贝、海参、章鱼和鱿鱼都属于一类海洋生物（软体动物）。本例使用来自 UCI 机器学习档案的鲍鱼数据集来预测鲍鱼的年龄（通过壳上环数），给定大量的属性，例如外壳尺寸（高度、长度、宽度）和重量（壳重、去壳重量、脏器重量、整个重量）。下面代码是如何使用 R 链接到数据集：

```
> aburl = "http://archive. ics. uci. edu/ml/machine - learning - databases/abalo
ne/abalone. data"
```

使用 read. table 加载数据并存储到 R 对象 data。

```
> names = c("sex","length","diameter","height","whole. weight",
            "shucked. weight","viscera. weight",
            "shell. weight","rings")
> data = read. table(aburl,header = F,sep = ",",col. names = names)
```

使用 summary 函数来考查不寻常的观测数据。

```
> summary(data)
```

```
sex       length           diameter          height            whole. weight
F: 1307   Min.    : 0.075  Min.    : 0.0550  Min.    : 0.0000  Min.    : 0.0020
I: 1342   1st Qu  : 0.450  1st Qu. : 0.3500  1st Qu. : 0.1150  1st Qu. : 0.4415
M: 1528   Median  : 0.545  Median  : 0.4250  Median  : 0.1400  Median  : 0.7995
          Mean    : 0.524  Mean    : 0.4079  Mean    : 0.1395  Mean    : 0.8287
          3rd Qu. : 0.615  3rd Qu. : 0.4800  3rd Qu. : 0.1650  3rd Qu. : 1.1530
          Max.    : 0.815  Max.    : 0.6500  Max.    : 1.1300  Max.    : 2.8255
shucked. weight   viscera. weight   she11. weight     rings
Min.    : 0.0010  Min.    : 0.0005  Min.    : 0.0015  Min.    : 1.000
1st Qu  : 0.1860  1st Qu. : 0.0935  1st Qu. : 0.1300  1st Qu. : 8.000
Median  : 0.3360  Median  : 0.1710  Median  : 0.2340  Median  : 9.000
Mean    : 0.3594  Mean    : 0.1806  Mean    : 0.2388  Mean    : 9.934
3rd Qu. : 0.5020  3rd Qu. : 0.2530  3rd Qu. : 0.3290  3rd Qu. : 11.000
Max.    : 1.4880  Max.    : 0.7600  Max.    : 1.0050  Max.    : 29.000
```

可以看出，鲍鱼高度存在问题；一些蜗牛高度为0，这是不可能的。进一步考查：

```
> data[data$height = =0,]
```

```
        sex   length   diameter   height   whole. weight
1258     I    0.430     0.34        0        0.428
3997     I    0.315     0.23        0        0.134
      shucked. weight   viscera. weight   shell. weight   rings
1258     0.2065           0.0860            0.1150          8
3997     0.0575           0.0285            0.3505          6
```

显现出了两个蜗牛高度为0 的观测数据（样本编号为1258 和3997）。需要删除这些观测样本。

```
> data$height [ data$height = = 0 ] = NA
> data <- na. omit( data)
> data$sex <- NULL      #去掉性别列
> summary( data)
```

```
    length          diameter          height          whole. weight
Min.    : 0.0750  Min.    : 0.0550  Min.    : 0.0100  Min.    : 0.0020
lst Qu. : 0.4500  lst Qu. : 0.3500  lst Qu. : 0.1150  lst Qu: : 0.4422
Median  : 0.5450  Median  : 0.4250  Median  : 0.1400  Median  : 0.8000
Mean    : 0.5241  Mean    : 0.4079  Mean    : 0.1396  Mean    : 0.8290
3rd Qu. : 0.6150  3rd Qu. : 0.4800  3rd Qu. : 0.1650  3rd Qu. : 1.1535
Max.    : 0.8150  Max.    : 0.6500  Max.    : 1.1300  Max.    : 2.8255
 Shucked. weight   Viscera. weight   shell. weight      rings
Min.    : 0.0010  Min.    : 0.0005  Min.    : 0.0015  Min.    : 1.000
lst Qu. : 0.1862  lst Qu. : 0.0935  lst Qu. : 0.1300  lst Qu. : 8.000
Median  : 0.3360  Median  : 0.1710  Median  : 0.2340  Median  : 9.000
Mean    : 0.3595  Mean    : 0.1807  Mean    : 0.2388  Mean    : 9.935
3rd Qu. : 0.5020  3rd Qu. : 0.2530  3rd Qu. : 0.3287  3rd Qu. : 11.000
Max.    : 1.4880  Max.    : 0.7600  Max.    : 1.0050  Max.    : 29.000
```

似乎数据都是合理的。接下来，转换数据，将它转换为一个矩阵，并将结果存储于 R 对象 data1。

```
> data1 <- t( data)
> data1 <- as. matrix(  data1  )
> require(  autoencoder  )
> set. seed( 2016)
> n = nrow( data)
> train <-  sample( 1 : n, 10, FALSE  )
> fit <- autoencode( X. train = data1 [ , train  ],
    X. test = NULL,
    nl = 3,
    N. hidden = 5,                #隐藏层结点个数
    unit. type = " logistic" ,
```

```
lambda = 1e - 5,
beta = 1e - 5,
rho = 0.07,
epsilon = 0.1,
max.iterations = 100,
optim.method = c("BFGS"),
rel.tol = 0.01,
rescale.flag = TRUE,
rescaling.offset = 0.001)
```

注意：有5个隐藏结点的模型和一个稀疏参数为7%。一旦模型被优化，均方误差小于2%。

```
> fit$mean.error.training.set
```

```
[1]0.01654644
```

通过设置 hidden. output = TRUE。由于隐藏结点的数目少于特征数目，feature$X. output 输出降维数据：

```
> features <- predict(fit,X.input = data1[,train],hidden.output = TRUE)
> features$X.output
```

	[,1]	[,2]	[,3]	[,4]	[,5]
length	6.572353e-01	7.459814e-01	4.025650e-01	5.306412e-01	6.325756e-01
diameter	7.149880e-01	7.907795e-01	4.670981e-01	5.899761e-01	6.929413e-01
height	8.119831e-01	8.628111e-01	5.986132e-01	7.009952e-01	7.956291e-01
whole. weight	4.901213e-01	6.072550e-01	2.545919e-01	3.770476e-01	4.642603e-01
shucked. weight	7.437931e-01	8.128396e-01	5.023596e-01	6.205615e-01	7.235000e-01
viscera. weight	7.893552e-01	8.464983e-01	5.645063e-01	6.734824e-01	7.718709e-01
shell. weight	7.721268e-01	8.337987e-01	5.403436e-01	6.532020e-01	7.534961e-01
rings	1.855435e-09	1.038285e-08	1.111354e-09	9.623515e-09	1.471309e-09

使用 predict 函数重构并存储结果到 R 对象 pred。

```
> pred <- predict(fit,X.input = data1[,train],hidden.output = FALSE)
```

图5.9、图5.10和图5.11使用雷达图可视化重建的值。

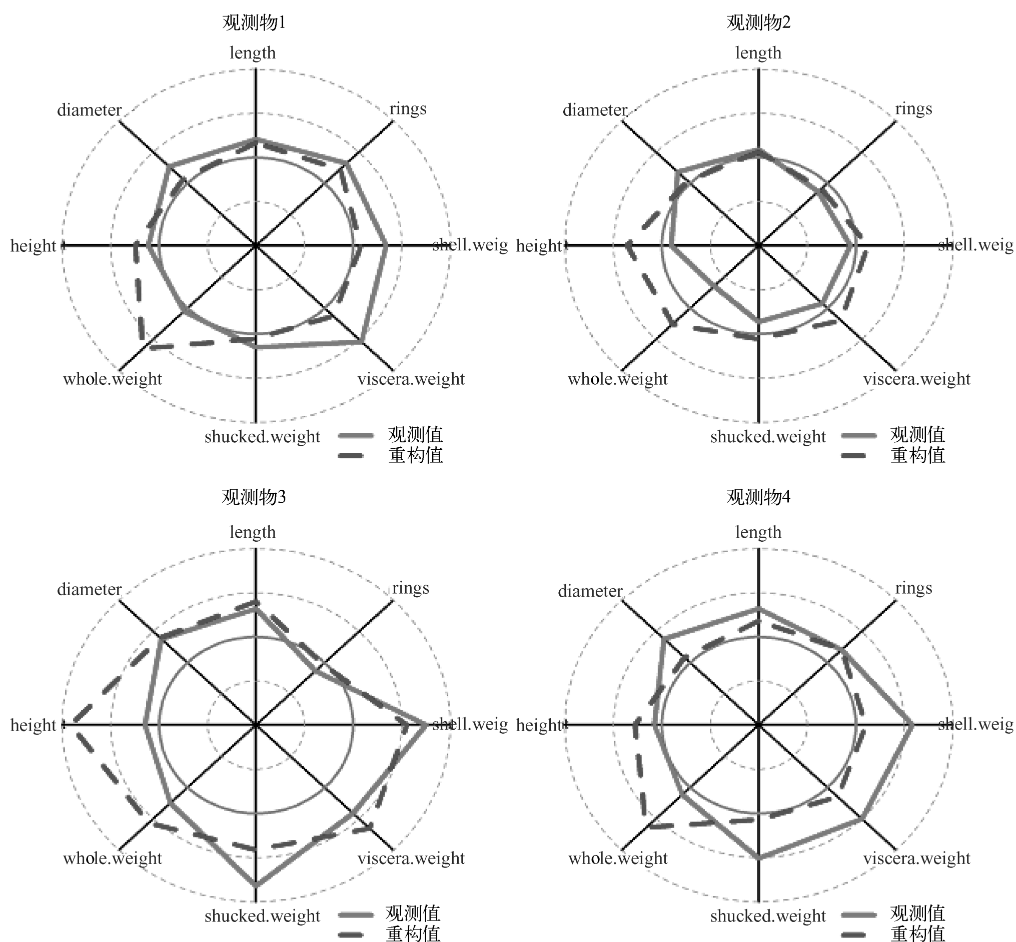


图 5. 9　对于观测物 1 ~ 4 所观测和重构的值

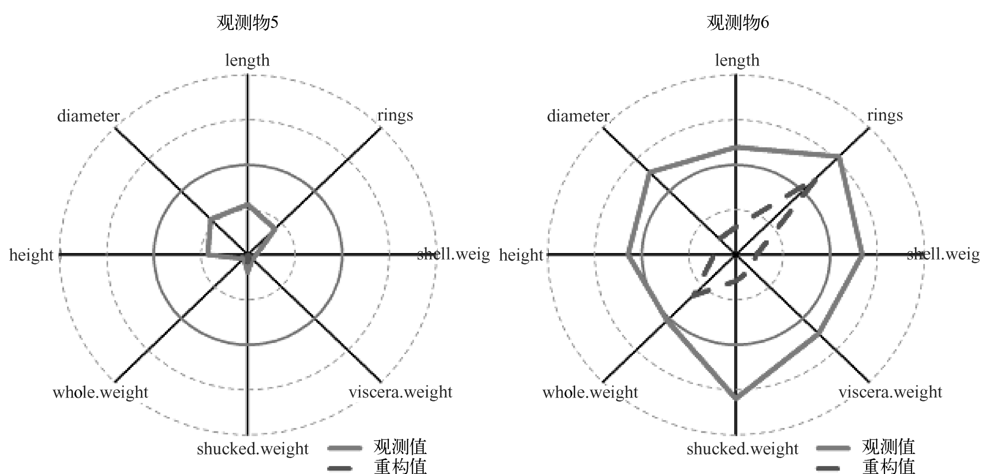


图 5. 10　观测物 5 ~ 8 所观测和重构的值

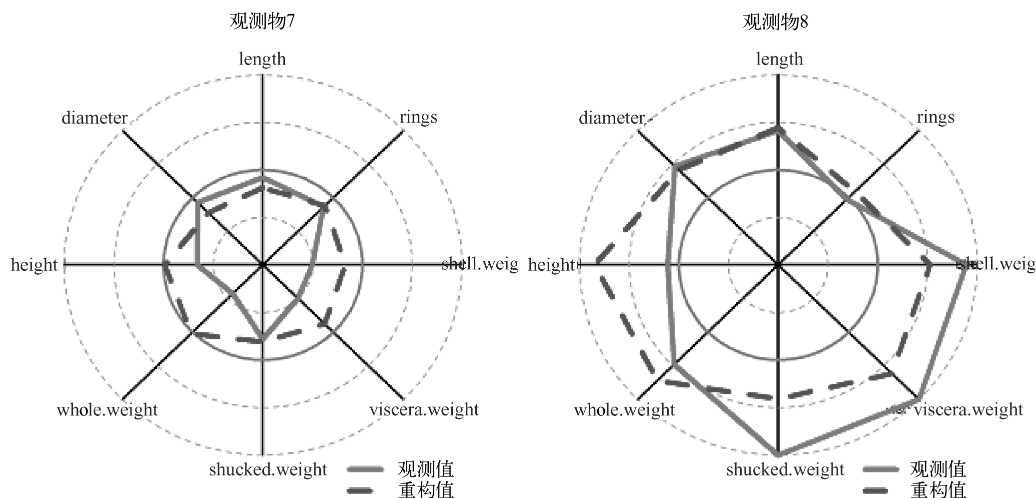


图 5.10　观测物 5 ~ 8 所观测和重构的值（续）

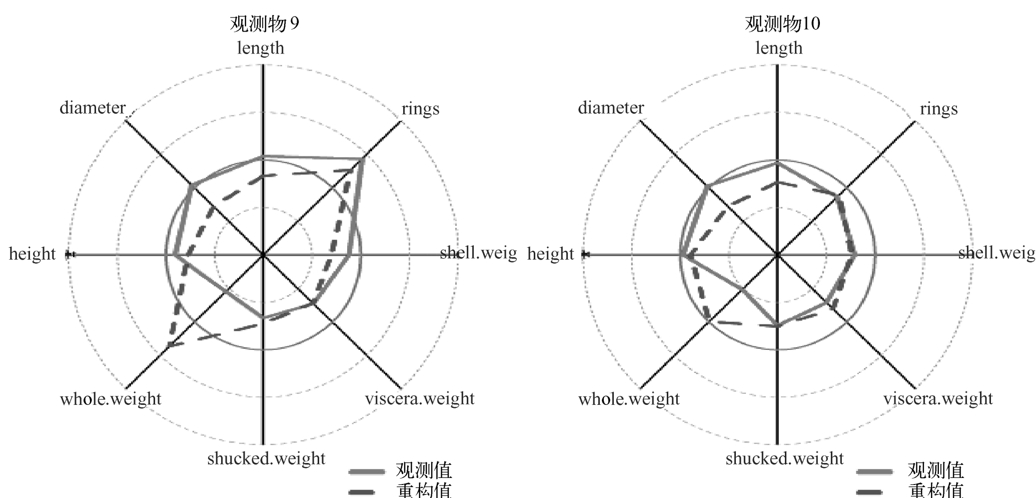


图 5.11　观测物 9 ~ 10 所观测和重构的值

整体重建的值提供了一个原始值的合理表达。注意，观测样本 5 拟合程度不如样本 6；进一步考查图 5.12 所示的柱状图，图中显示了重建值和观测值之间的差异。重建值在所有八个维度（属性）under – fit 观测值。在观测样本 5 最明显的是环，重建环的值为 1.4，相对所观测到的值为 3.5。观测样本 6 观测到一个类似的模式。

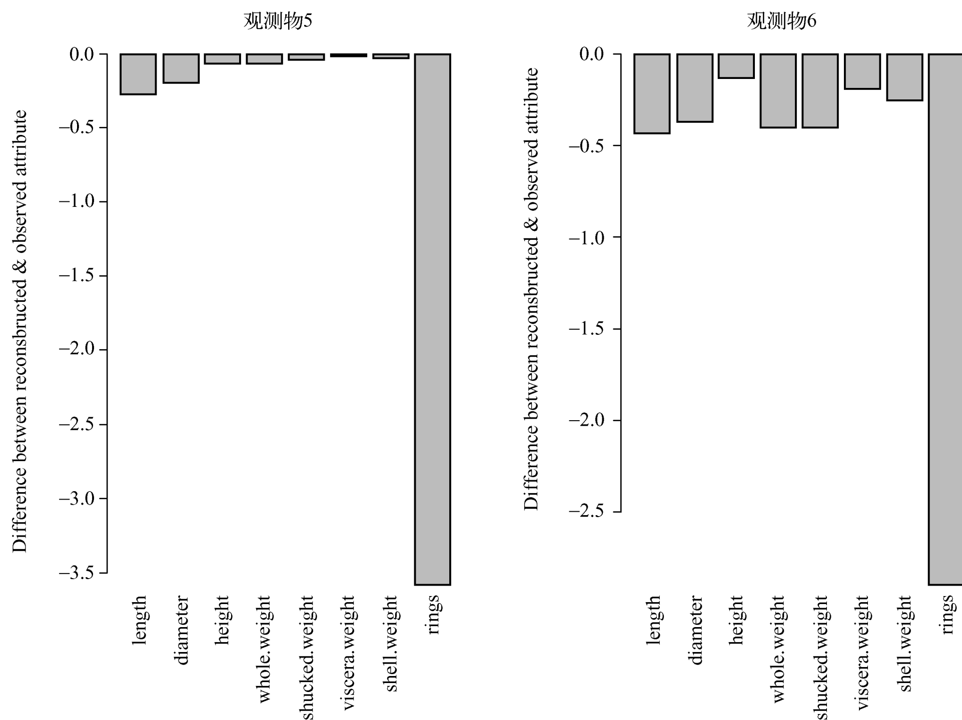


图 5.12　观测物 5、6 的柱状图

5.5　学习指南

大多数介绍机器学习课程倾向于在前馈神经网络停止了。其实可拓展的空间还很多。

一个 AE 通常是一个前馈神经网络，目的是学习一个压缩、分布式的数据集表示（编码）。

从概念上讲，AE 网络被训练为输入的“再现”，输入和目标数据是相同的。

AE 网络有多种变体：

Sparse AE（稀疏自编码器）

Denoising AE（降噪自编码器）

Regularized AE（正则自编码器）

Contractive AE（具有惩罚项的 AE）

Marginalized DAE（边际降噪自编码器）

第 6 章　堆栈自编码网络 SA

有了 AE 的基本概念，可以利用 AE 来构建深度网络，近些年的一系列研究表明构建深度网络对于解决很多的计算机视觉问题具有重要意义，并能比现有的一些常规方法取得更好的效果。

堆栈自编码（Stacked Autoencoder，SA）网络，或层叠自编码网络就是一种利用 AE 来构建深度网络的一种策略。深度网络的每一层都是一个编码器，在这个编码器中，每一层的输出连接到下一层的输入。为了紧凑地表示，中间层中单元的数量往往变得越来越小。

图 6. 1 是出自 Jirayucharoensak 等发表论文的 SA 网络，它是拥有两个隐藏层的二分类堆栈自编码网络。在这个例子里，向堆栈自编码网络中输入四个属性 x_1、x_2、x_3、x_4。中心带有 +1 标志的结点表示偏置（Bias）。属性数据穿过两个隐藏层，每一个隐含层含有三个结点。输出层由一个 Softmax 激活函数构成，激活函数将计算出该输出属于某一具体类的概率（此案例中为 0 或 1）。注意，分类时，通常 Softmax 层中每一类只有一个输出结点。

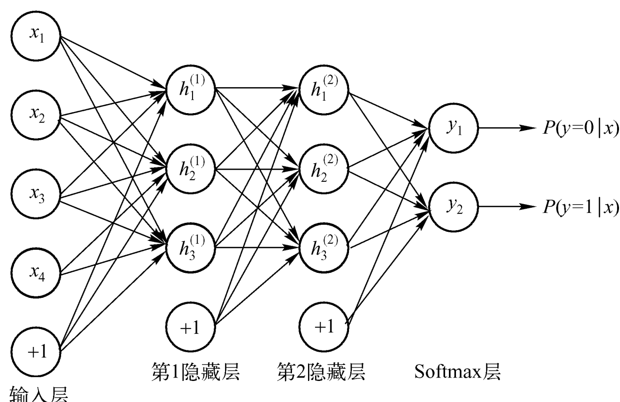


图 6. 1　拥有两个隐藏层的二分类堆栈自编码网络

6.1　SA 原理

（1）逐层训练

图 6.1 第一层得到一个特征，如果重构误差小于给定的阈值，可以确定这个特征就是原输入信号的良好表达，或者牵强点说，它和原信号是一模一样的（表达不一样，反映的是一个东西）。那第二层和第一层的训练方式就没有差别了，将第一层输出的特征当成第二层的输入信号，同样最小化重构误差，就会得到第二层的参数，并且得到第二层输入的特征，也就是原输入信息的第二个表达（见图 6.2）。其他层采用同样的方法（训练这一层，前面层的参数都是固定的，并且解码器已经没用了，都不需要了）。

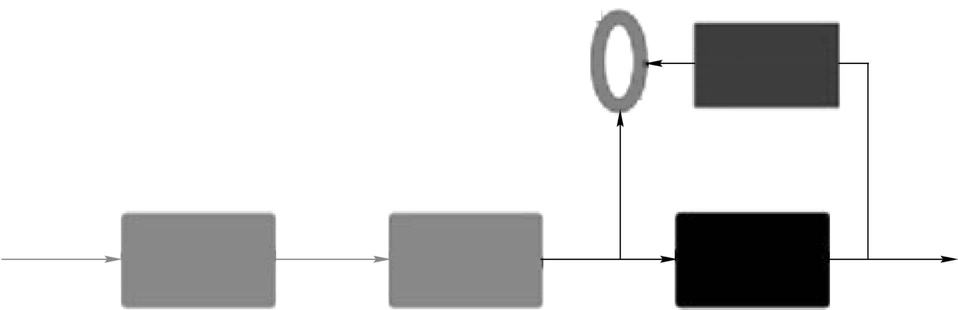


图 6.2　逐层训练过程

（2）微调

经过逐层训练，需要自己调试这个网络。每一层都会得到原始输入的不同表达。当然，越抽象（层数越多）越好，就像人的视觉系统一样。

到这里，这个 SA 还不能用来分类数据，因为它还没有学习如何去联结一个输入和一个类。它只是学会了如何去重构或者复现它的输入而已。或者说，它只是学习获得了一个可以良好代表输入的特征，这个特征可以最大程度上代表原输入信号。那么，为了实现分类，可以在 AE 的最顶的编码层添加一个分类器（例如 Logist 回归、SVM 等），然后通过标准的多层神经网络的监督训练方法（梯度下降法）去训练。

也就是说，这时候，需要将最后层的特征输入到最后的分类器，通过有标签样本和监督学习进行微调，这也分两种，一种是只调整分类器（黑色部分，见

图 6.3）；另一种通过有标签样本，微调整个系统（如果有足够多的数据，这个是最好的）。

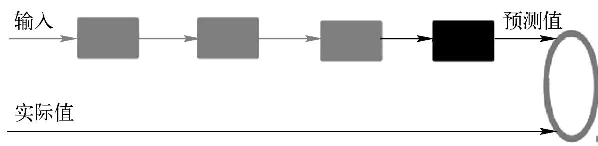


图 6.3　微调过程

一旦有监督训练完成，这个网络就可以用来分类了。神经网络的最顶层可以作为一个线性分类器，然后用一个更好性能的分类器去取代它。

在研究中可以发现，如果在原有的特征中加入这些自动学习得到的特征可以大大提高精确度，甚至在分类问题中比目前最好的分类算法效果还要好！

在上述的堆栈自编码网络用于传统训练方法的流程的时候，一些研究人员对预训练这一步的必要性提出了疑问。但实验告诉我们预训练是最好的建议。

6.2　SA 的 R 实现

【例 6.1】 使用 SAENET 包的 SAENET. train 函数对 SA 模型进行评估。

（1）加载所需包和数据

```
> require(SAENET)
> aburl = 'http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'
> names = c('sex','length','diameter','height','whole.weight',
    'shucked.weight','viscera.weight','shell.weight','rings')
> data = read.table(aburl,header = F,sep = ',',col.names = names)
```

（2）数据准备

去掉性别属性，删除编码出错率高的观测值，并且将结果样本存储为 R 中矩阵对象 data1。

```
> data$sex <- NULL                    #去掉性别属性
> data$height[data$height == 0] = NA
> data <- na.omit(data)
> data1 <- as.matrix(data)
```

为了说明问题，只抽取10个观测值。

```
> set. seed(2016)
> n = nrow(data)
> train <=  sample(1:n,10,FALSE)
```

此时，对模型的评估已准备就绪。在这个例子中，设计3个隐藏层，且结点数分别为5、4、2的模型{n. nodes = c(5,4,2)}。余下的代码把模型存储到R中fit对象。

```
> fit <= SAENET. train(X. train = data1[train,],
n. nodes = c(5,4,2),
unit. type = "logistic",
lambda = 1e  -5,
beta = 1e  -5,
rho = 0. 07,
epsilon = 0. 1,
max.  iterations = 100,
optim. method = c("BFGS"),
rel. tol = 0. 01,
rescale. flag = TRUE,
rescaling. offset = 0. 001)
```

每一层的输出可以通过输入fit[[n]]$X. output来查看，这里n是感兴趣的层（the Layer of Interest）。例如，查看第三层的两个结点的输出：

```
> fit[[3]] $X.  output
          [,1]        [,2]
753    0. 4837342   0. 4885643
597    0. 4837314   0. 4885684
3514   0. 4837309   0. 4885684
558    0. 4837333   0. 4885653
1993   0. 4837282   0. 4885726
506    0. 4837351   0. 4885621
2572   0. 4837315   0. 4885684
3713   0. 4837321   0. 4885674
```

11	0.4837346	0.4885632
223	0.4837310	0.4885684

图 6.4 绘制了观测的输出值，为了清晰起见，将第 753 项标注为观测值 1，第 597 项标注为观测值 2，…，第 223 项标注为观测值 10。

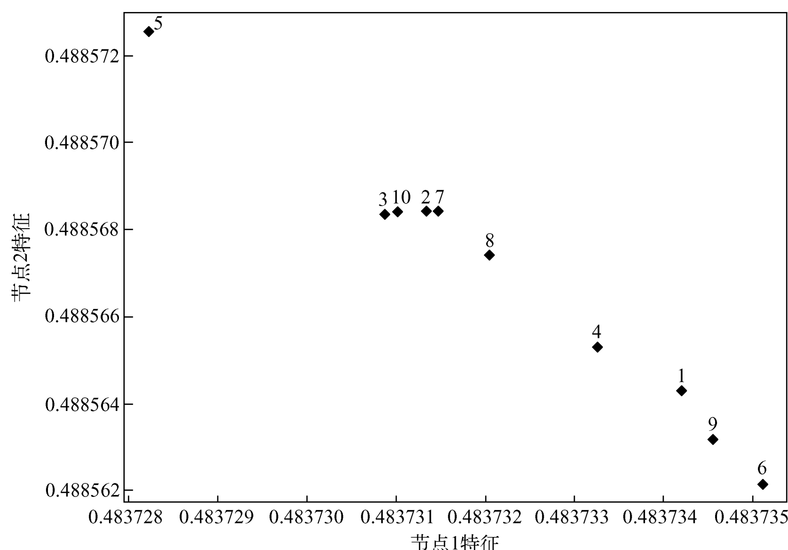


图 6.4　第三层特性的可视化表示

6.3　降噪自编码网络 DAE

降噪自动编码（Denoising AutoEncoders，DAE）网络是在自动编码网络的基础上，对训练数据加入噪声，是 AE 的变体，所以自动编码网络必须学习去除这种噪声而获得真正的没有被噪声污染过的输入。因此，这就迫使编码器去学习对输入信号更加鲁棒的表达，这也是它的泛化能力比一般编码器强的原因。DAE 可以通过梯度下降算法训练。其结构如图 6.5 所示。

本节的降噪自编码网络与传统的降噪自编码网络的区别是，所添加的噪声可以预防隐藏层对恒等函数简单地学习，强制它发掘更好的特性。

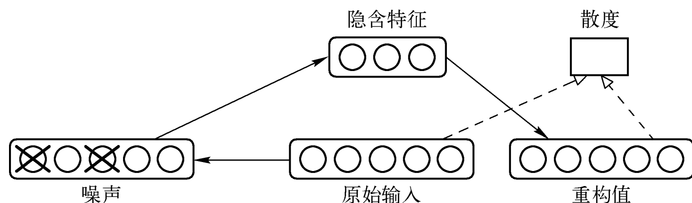


图 6.5　降噪自动编码网络结构

6.3.1　随机掩蔽的椒盐噪声

给定输入属性 $\boldsymbol{x}$，降噪自编码网络会产生一个新的“嘈杂”的属性向量$\tilde{\boldsymbol{x}}$。虽然有很多方法引入噪声，其中最常用的是随机掩蔽。在随机掩蔽方法中，输入是随机的，一般设定为 0。注意，这一想法本质是在每轮的训练中随机地忽略一些神经元，其性能提高明显。通常把随机强制将 $\boldsymbol{x}$ 的一小部分转化成 0 或 1 称为附加椒盐噪声的变体；另一种常用方法是向属性向量 $\boldsymbol{x}$ 附加高斯噪声。不管使用哪种噪声方法，接下来都要训练降噪自编码网络，重建来自不同噪声版本的输入属性。

6.3.2　DAE 基本任务

降噪自编码网络有两个基本任务：

1）对输入属性向量 $\boldsymbol{x}$ 进行编码。

2）去除噪声属性向量 $\tilde{\boldsymbol{x}}$ 内的错误所造成的影响。

只要降噪自编码网络捕获输入属性之间的统计特征依赖，这些任务就能成功完成。在实践中，一个降噪自编码网络通常会比标准自编码网络获得更好的输入属性表示方法。

6.3.3　标准化堆栈降噪自编码网络

降噪自编码网络可以通过堆栈形成深度学习网络。图 6.6 展示了一个有 4 个隐藏层的堆栈降噪自编码网络（SDA）。

如堆栈自编码网络一样，在堆栈降噪自编码网络中，一次只在一层用同样的方式进行预训练。在反向传播阶段，预训练经常激励网络去发掘更好的参数空间，随时间推移，这一现象会变得更加清晰。

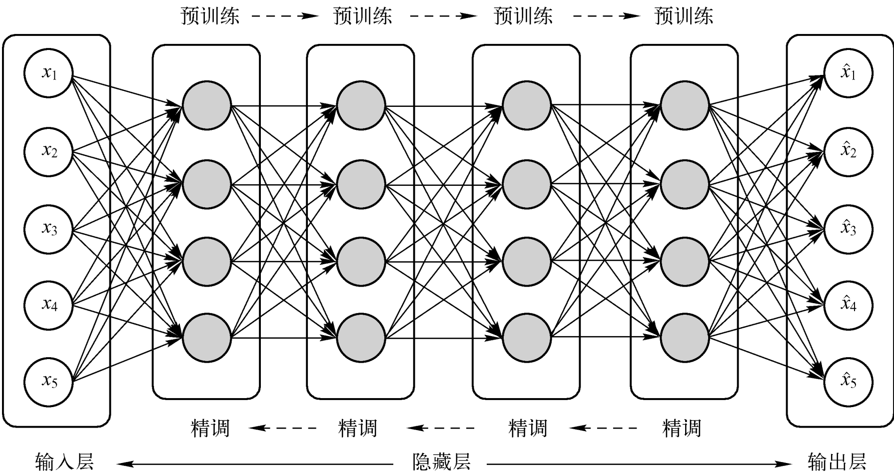


图 6.6　由 4 个隐藏层构造的堆栈降噪自编码网络

在预训练阶段，通过最小化重建的错误来将每一层训练成一个降噪自编码网络。对于每一层来说，用将要输入到下一个隐藏层的权重表示提取的特征。在最后一个预训练之后，通常用 Softmax 激励函数将结果向量传递到输出层来对最后一个隐藏层进行分类。

一旦所有的层都进行了预训练，网络就进入第二阶段的训练，即有监督的学习，通常称为微调。在这一阶段，通过反向传播，将整个网络像训练所得的多层感知器那样最小化预测错误。

堆栈降噪自编码网络似乎提高了网络的表示能力。添加的层数越多，特征提取得就越抽象。

6.4　DAE 的 R 实现

作为对孟加拉国妇女生育率定期调查的一部分，Huq 和 Cleland 收集了社会自由流动性数据。他们对 8445 名农村妇女是否可以单独从事某些活动进行了问卷调查(见表 6.1)。

表 6.1　妇女的社会自由流动性所使用的变量

项目编号	内　　容
1	去过村里任何地方
2	去过村外

（续）

项目编号	内　　容
3	与陌生男人谈话
4	去影院看电影或剧院看戏
5	购物
6	去过母亲俱乐部
7	出席政治会议
8	去过健康中心或医院

【例 6. 2】使用这个示例数据构建自编码网络。

（1）加载依赖的包和数据

```
> require(RcppDL)
> require("ltm")
> data(Mobility)
> data <= Mobility
```

（2）数据准备

从 8445 个样本中不重复抽取了 1000 个样本观测值。其中 800 个观测值用作训练集，剩下 200 个观测值用作测试集。

```
> set. seed(17)
> n = nrow(data)
> sample <=  sample(1:n,1000,FALSE)
> data <= as. matrix(Mobility[sample,])
> n = nrow(data)
> train <=  sample(1:n,800,FALSE)
```

创建训练样本和测试样本代码如下：

```
> x_train <= matrix(as.  numeric(unlist(data[train,])),nrow = nrow(data[train,]))
> x_test <= matrix(as.  numeric(unlist(data[ -train,])),nrow = nrow(data[ -train,]))
```

需要进行检查以确保有正确的样本容量。训练集应该为 800 个观测值，测试集应该有 200 个观测值。

```
> nrow(x_train)
[1] 800
```

```
> nrow( x_test)
[1] 200
```

所有这些看起来都还不错。现在把原来的响应变量从属性 R 对象中删除。在这个例子中，将 Item3（与一个不了解的男人说话）作为响应变量。下面代码示例怎样将Item3从属性对象中删除：

```
> x_ train <= x_train[ , -3 ]
> x_test <= x_test[ , -3 ]
```

（3）建模

接下来使用 RcppDL 包中的 Rsda 函数构造降噪自编码网络。需要用两个响应变量。首先为训练样本创建响应变量。

```
> y_train <= data[ train,3 ]
> temp <= ifelse( y_ train ==0,1,0)
> y_ train <= cbind( y_train,temp)
```

对于测试样本，遵循相同的程序。

```
> y_test <= data[ -train,3 ]
> temp1 <= ifelse( y_test ==0,1,0)
> y_test <= cbind( y_test,temp1 )
```

现在来详细说明模型。以上构造了一个没有任何噪声的堆栈自编码网络，两个隐藏层，每层包含 10 个结点。

```
> hidden = c( 10,10)
> fit <= Rsda( x_train,y_train,hidden)
```

Rsda 默认的噪声等级是 30% 。因为是从一个常规的堆栈自编码网络开始的，所以将噪声设置为 0。

```
> setCorruptionLevel( fit,x =0. 0)
```

注意：可以在 Rsda 包中设置很多参数。

```
> setCorruptionLevel( model,x)
```

也可以为微调和预训练阶段选择样本数量和学习率。

```
setFinetuneEpochs
```

setFinetuneLearningRate

setPretrainLearningRate

setPretrainEpochs

预训练和微调模型相当简单。

```
> pretrain(fit)
> finetune(fit)
```

（4）模型部署

因为样本很小，所以模型收敛很快。可以使用测试样本来看看对响应变量预测的概率。

```
> predProb <= predict(fit,x_test)
> head(predProb,6)
```

```
          [,1]        [,2]
[1,]  0.4481689  0.5518311
[2,]  0.4481689  0.5518311
[3,]  0.4481689  0.5518311
[4,]  0.6124651  0.3875349
[5,]  0.4481689  0.5518311
[6,]  0.8310412  0.1689588
```

观察到模型预测的前三个观测值中有 45% 属于 1 类，55% 属于 2 类。来看看它是怎样实现的。

```
> head(y_test,3)
```

```
      y_test  temp1
3954    1       0
1579    0       1
7000    0       1
```

虽然第一个观测错过了！然而，第二个和第三个观测值正在被正确地分类。最后，构建混淆矩阵。

```
> pred1 <= ifelse(predProb[,1] >= 0.5,1,0)
> table(pred1,y_test[,1],dnn = c("Predicted"," Observed "))
```

```
            Observed
```

```
Predicted   0    1
    0       15   15
    1       36   134
```

接下来，重建模型。这一次添加 25% 的噪声。

```
> setCorruptionLevel(fit,x = 0.25)
> pretrain(fit)
> finetune(fit)
> predProb <= predict(fit,x_test)
> pred1 <= ifelse(predProb[,1] >= 0.5,1,0)
> table(pred1,y_test[,1],dnn = c("Predicted","Observed"))
            Observed
Predicted   0    1
    0       15   15
    1       36   134
```

这个混合矩阵与没有任何噪声的堆栈自编码网络的一样。所以，在这种情况下添加噪声没有多少好处。

6.5 学习指南

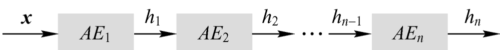
有过深度学习基础的读者想必了解，深层网络的威力在于其能够逐层地学习原始数据的多种表达。每一层的表达都以低一层的表达为基础，但往往更抽象，更加适合复杂的分类等任务。

堆栈自编码器实际上就在做这样的事情，如前所述，单个自编码器通过虚构 $\boldsymbol{x}\rightarrow\boldsymbol{h}\rightarrow\boldsymbol{x}$ 的三层网络，能够学习出一种特征变化 $\boldsymbol{h}=f_{\theta}(\boldsymbol{x})$（这里用 θ 表示变换的参数，包括 $\boldsymbol{W}$、b 和激活函数）。实际上，当训练结束后，输出层已经没什么意义了，一般将其去掉，即将自编码器表示为



AE 之所以将自编码器模型表示为 3 层的神经网络，是因为训练的需要，将原始数据作为假想的目标输出，以此构建监督误差来训练整个网络。等训练结束后，输出层就可以去掉了，只关心是从 $\boldsymbol{x}$ 到 $\boldsymbol{h}$ 的变换。

接下来的思路就很自然了——对已经得到的特征表达 $\boldsymbol{h}$，当作新的原始信息，再训练一个新的自编码器，得到新的特征表达。这就是所谓的堆栈自编码器（Stacked Auto encoder，SA）。Stacked 就是逐层垒叠的意思，跟“栈”有点像。UFLDL 教程将其翻译为“栈式自编码”，不管怎么称呼，意思是一样的。当把多个自编码器 Stack 起来之后，这个系统看起来就像这样：



这个系统实际上已经有点深度学习的味道了。需要注意的是，整个网络的训练不是一蹴而就的，而是逐层进行。按上面提到的结构 n，m，k，实际上是先训练网络 $n \rightarrow m \rightarrow n$，得到 $n \rightarrow m$ 的变换，然后再训练 $m \rightarrow k \rightarrow m$，得到 $m \rightarrow k$ 的变换。最终堆叠成 SA，即为 $n \rightarrow m \rightarrow k$ 的结果，整个过程就像一层层往上盖房子，这便是知名的 Layer - wise Unsuperwised Pre - training（逐层非监督预训练），这正是导致深度学习在 2006 年第 3 次兴起的核心思想。

第 7 章　受限玻耳兹曼机 RBM

RBM 是玻耳兹曼机（Boltzmann Machine，BM）的一种特殊拓扑结构。BM 的原理起源于统计物理学，是一种基于能量函数的建模方法，能够描述变量之间的高阶相互作用。BM 的学习算法较复杂，但所建模型和学习算法有比较完备的物理解释和严格的数理统计理论作基础。BM 是一种对称耦合的随机反馈型二值单元神经网络，由可见层和多个隐层组成，网络结点分为可见单元和隐单元，用可见单元和隐单元来表达随机网络与随机环境的学习模型，通过权值表达单元之间的相关性。

受限玻耳兹曼机（Restricted Boltzmann Machine，RBM）是一种无监督学习模型，近似于样本数据的概率密度函数。名字中“受限”一词指的是相同层的单元之间没有连接。模型的参数由样本的最大化似然函数学习得到。因为它是用来逼近概率密度函数的，所以在研究领域通常称为生成模型。生成模型需要猜测原始输入的概率分布，以便于能够重建它。

7.1　RBM 原理

7.1.1　玻耳兹曼机的四类知识

图 7.1 展示了受限玻耳兹曼机的几何表示。它由两层构成，在这两层中有很多与内层连接的单元。层之间的连接是对称和双向的，这允许两个方向传递信息。它有一个包含 3 个结点的可见层和一个包含 4 个结点的隐藏层，可见结点与输入属性有关，因此对可见层的每个单元来说相关属性的值是可观测的。

为说明受限玻耳兹曼机工作原理，下面考虑所有结点（神经元）是二进制的常用情况。这种情况下理解玻耳兹曼机需要四类知识：

1）可见结点的取值为 $v_i=0$ 或 $v_i=1$。

2）每个结点都是一个处理输入的计算轨迹，并随机决定传递哪个输入。

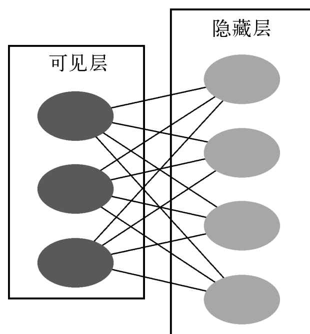


图 7.1　受限玻耳兹曼机的几何表示

3）隐藏单元用于增加模型的可表达性和容纳 $h_j=0$ 或 $h_j=1$ 的二进制单元。

4）对于隐藏层中的每个单元或结点相应的值是不可观测的，需要推断。

7.1.2　能量和概率的作用

受限玻耳兹曼机是一个基于能量的概率模型，这就意味着可见层和隐藏层单元的具体配置的概率与能量函数$\widetilde{E}(v,h)$的负幂次方成正比。对于每一组可见向量和不可见向量来说，这组(v,h)的概率定义为 $P(v,h)=\frac{1}{Z}\exp(-\widetilde{E}(v,h))$，其中分母 Z 是标准化常数，称为赋值函数。

对成对的可见和隐藏变量求和的赋值函数计算如下：$Z=\sum_v\sum_h\exp(-\widetilde{E}(v,h))$，因此，它是 $P(v,h)$对所有 v 和 h 定义的一个概率分布。

注解：

一个由两部分构成的结构或图是对一组结点的数学表示，这些结点只能是两种颜色中的一种，并用线（边）连接结点（顶点），以至于不会出现一条线连接两个同种颜色的结点。图 7.2 举例说明了这一点。

在受限玻耳兹曼机中，这种由隐含层和可见层结点两部分构成的结构图如图 7.1 所示。

因为一个受限玻耳兹曼机内没有层内连接，所以定义由两部分构成的联合配置(v,h)的能量如下：$\widetilde{E}(v,h)=-\sum_{i=1}^{m}\sum_{j=1}^{n}w_{ij}v_ih_i-\sum_{i=1}^{m}v_ib_i-\sum_{j=1}^{n}h_jd_j$，这里 w_{ij}是与 v_i 和 h_j之间的连接相关的权重。直观地说，权重确定结点的相关性。大的权重意味着连接的结点一致的可能性大。参数 b_i和 d_j是隐藏层的第 j 个结点和第 i 个可见层结点的相对偏置；m 和 n 是可见层和隐藏层的结点数。

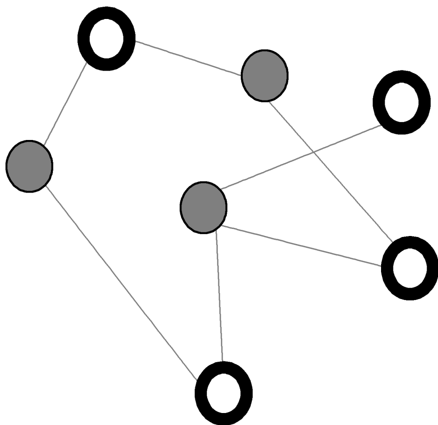


图 7.2　由两部分构成的结构

7.1.3　联合概率分布表示的自编码网络

另一种对受限玻耳兹曼机启发的是可见层结点和隐藏层结点的联合概率分布的参数模型。它是一种用于学习一组数据的联合概率分布表示（编码）的自编码网络。由于受限玻耳兹曼机的隐藏层单元只与它们外面可见层连接，所以隐藏层与可见层是相互独立的。同层神经元相互独立是一种很好的性质，因为它允许在给定可见层单元的条件下，对隐藏层单元的条件分布 $P(h \mid v) = \prod_j P(h_j \mid v)$ 进行如下因式分解：

$$P(h_j = 1 \mid v) = \prod_j sigmoid\left(\sum_i w_{ij} v_i + d_j\right)$$

可见层单元的条件分布 $P(v \mid h) = \prod_i P(v_i \mid h)$ 可以用分解的因式乘积来表示：

$$P(v_i = 1 \mid h) = \prod_i sigmoid\left(\sum_j w_{ij} v_j + b_i\right)$$

在训练受限玻耳兹曼机时，这些条件概率对于隐含层和可见层之间的迭代更新非常重要。在实践中，Z 的计算非常困难，因此联合概率分布 $P(v,h)$ 的计算通常很棘手。

要注意，受限玻耳兹曼机有两个偏置，这是它与其他自编码网络的区别。隐藏偏置 d_j 有助于受限玻耳兹曼机产生向前穿过网络的激励，而可见层偏置 b_i 则帮助受限玻耳兹曼机在反向传播或反向经过网络时学习重建。

7.1.4　模型学习的目标

训练受限玻耳兹曼机的目标是调整模型的参数，用 Θ 表示训练参数，对使用训

练数据的对数似然函数最大化。

对数似然函数的梯度是 $\frac{\partial}{\partial \Theta}L(\Theta) = -\left\langle \frac{\partial \widetilde{E}(v;\Theta)}{\partial \Theta} \right\rangle_{\text{data}} + \left\langle \frac{\partial \widetilde{E}(v;\Theta)}{\partial \Theta} \right\rangle_{\text{model}}$。这里 $\langle \cdot \rangle_{\text{data}}$ 代表所有关于数据分布的可见结点 v 的期望；$\langle \cdot \rangle_{\text{model}}$ 表示由 $P(v,h)$ 定义的模型的联合分布。

不幸的是，对数似然函数梯度很难计算，为了解决这一问题，需要一些训练技巧。

7.2 训练技巧

7.2.1 技巧1：Gibbs 采样

第一个训练技巧需要用到 $P(h \mid v)$ 和 $P(v \mid h)$ 的表达式。既然它们两者的表达式是容易处理的，为什么不使用它们通过 Gibbs 采样对联合分布 $P(v,h)$ 进行采样呢？

Gibbs 采样是从多维随机分布的联合分布中产生样本的一种蒙特卡罗马尔可夫链算法。其基本思想是在变量条件分布给出其他变量状态的基础上，通过更新每个变量构建马尔可夫链。

在 Gibbs 采样中，在给隐藏结点赋定值的时候对可见结点采样。同样，在给可见结点赋定值的时候对隐藏结点采样。对于马尔可夫链中的每一步，都会随机抽取隐藏层单元，并根据它们 Sigmoid 函数确定的概率赋值为 1 或 0。

随着步数趋近无穷大，隐藏层和可见层结点的采样会收敛于联合分布 $P(v,h)$。通过这个技巧，梯度的估计就很容易获得了。

因为交替的 Gibbs 采样的一次迭代需要为 $P(h \mid v)$ 并行更新所有的隐藏层单元，紧接着为 $P(v \mid h)$ 并行更新所有的可见层单元。所以 Gibbs 采样可能会花费很长时间来收敛，这取决于对速度的需求。

7.2.2 技巧2：最小化 KL 距离

最小化对比散度 Kullback – Leibler 距离。这里不介绍具体细节，但是核心是通过设定一个训练向量的可见单元的状态启动马尔可夫链。然后隐藏单元的二进制状态都是通过并行计算得到，用于计算 $P(v \mid h)$。一旦从隐藏单元选定了二进制状态，

按照 $P(v \mid h)$ 所给的概率将每个 v_i 设定为 1，实现了“重建”。现在，只对很小数量的迭代进行 Gibbs 采样，不需要等待收敛。

如果用 k 表示 Gibbs 采样的迭代的数量，很多应用都令 $k=1$，因为 1 远小于无穷，这节省了很多时间。总之，从数据样本初始化的 Gibbs 链采样 k 步就会产生对比散度。

在实践中，更高效的做法是一次更新多个训练样本，对产生的更新取平均。这样就平滑了学习信号，并充分利用了大矩阵运算的效率。

使用 Gibbs 采样，对观测数据和自由能量函数可以粗略估计对数似然函数梯度值。梯度下降算法与反向传播机制的结合使对数似然函数估计值 $\hat{\Theta}$ 的获取更高效。

7.2.3 技巧 3：使用 RLU 激活函数

标准的受限玻耳兹曼机使用 Sigmoid 激活函数，在可见层和隐藏层使用二进制单元（BU）。但是，也可以使用很多其他的激活函数，实际上，输出范围在［0，1］的实数可以用 Logistic 激活函数和没有做任何修正的对比散度作为训练模型。

另一种流行的激活函数是修正的线性单元（RLU），它展现了强大的图像分析能力。在一个使用 32 ×32 彩色图像任务中，相比 BU 的 0.7777 预测精度，RLU 的预测精度为 0.8073。然而，对于散度来说，大的标准差具有统计学意义。

RLU 的另一个优势是，它的参数不多于 BU，但它却更具表达力。正如 Vinod 和 Hinton 所陈述的“与二进制单元相比，这些修正的线性单元学习的特征在 NORB 数据集上能更好地进行目标识别，在 Wild 数据集中能很好地完成脸部认证”。与 BU 不同，RLU 保存了相关强度有关的信息，这些信息经过多个层次的特征探测。

7.2.4 技巧 4：模拟退火

模拟退火是一种对比散度的替代方案。它使用另一种对 $P(v,h)$ 的抽样的近似，$P(v,h)$ 依赖于一条有持续状态的马尔可夫链。在这一持续状态中，参数更新后不使用数据样本初始化马尔可夫链。在最后一步的更新中使用最后的链状态。如果学习

效率选择得足够小，这项技术将会带来更好的梯度近似值。

模拟退火通过引入对参数的额外设置来尝试提高学习速度，这种额外的参数设置只用于学习期间的 Gibbs 采样过程。模拟退火的想法是通过引入补充的 Gibbs 链，模拟原始系统的副本。

通常每个样本的温度不同。温度越高，对应链的分布越平滑。并行回火通过允许系统在不同的温度，根据 Metropolis Hastings 比率给出的概率，交换配置使链混合得更好（取得良好的抽样）。原来 K 个副本的仿真超过 $1/K$ 次，比标准的和单一温度的蒙特卡罗仿真更高效。这一技术的另一个优点是可以高效地利用大型 CPU 集群，在 CPU 集群中，不同的副本可以并行地运行。

7.3 对深度学习的质疑

为什么对比散度方法会奏效？至今理论学家也不能给出满意的答案。Sutskever 和 Tieleman 指出，学习规则不遵循任何函数梯度，尽管对比散度取得了经验上的成功，但是对它的收敛性质的理论知之甚少。波士顿大学的物理学 Pankaj Mehta 教授和西北大学生物物理学 David Schwab 教授就深度学习指出，“尽管深度学习取得了巨大成功，但很少有人在理论上去探究为什么深度学习技术在特征学习和压缩上会如此成功。”

对比散度就不是很好理解，但这并不是问题，最成功的数据科学家是务实的，他们先于理论家提出解决实际问题的创新方案。如果一项技术非常好用，数据学家就会一直使用它，直到发现更好的技术。对比散度的方法非常好以至于可以在很多重要的应用中取得成功。

一旦理论学家迎头赶上，理论会为为什么这一想法是个伟大的想法提供额外的技术支持。这是一个非常重要的行动，但是我们不能等待理论的成熟，继续奔跑吧！

7.4 RBM 应用

在过去的 10 年中，RBM 已经广泛应用于很多的应用中，包括降维、分类、协同过滤、特征学习和主题建模。本节来探究这方面的两个应用。

7.4.1　肝癌分类的 RBM

临床研究人员已经得出结论，癌症血清含有一种与一组独特的自体反应细胞抗原起反应的抗体，这种抗原称为肿瘤相关抗原。这是一个激动人心的消息，因为微型抗原阵列有探测和诊断各种类型的癌症的潜力。技巧在于将抗原与特定类型的抗体正确地联合。一旦这一微型阵列被创建，这就变成了适合受限玻耳兹曼机的分类问题。

Koziol 等人研究了用于肝癌分类的 RBM，分类的标准是 Logistic 回归。Logistic 回归是线性回归的分支，广义线性模型家族的成员之一，几十年来在医学统计领域拥有至高无上的统治权。

因为 Logistic 回归是基于结果概率进行建模的，所以它能完全适应分类任务，并在许多学科中赢得了金标准的地位。Logistic 回归是使用最广泛的二进制分类数据分析模型之一。

Koziol 等人将收集的 175 名肝癌患者和 90 名正常名人的血清样本用于 Logistic 回归和受限玻耳兹曼机并进行比较研究，12 种抗原表达为重组蛋白。研究人员分别对受限玻耳兹曼机模型和 Logistic 回归模型用 10 倍交叉验证。整个数据集随机分为 10 个同样大小的子样本。每个子样本被分层以保护癌症病例的比例控制。用 9 个子样本对 Logistic 回归和受限玻耳兹曼机进行训练；第 10 个样本用于确认。

表 7.1 中展示了使用 12 种抗原的二分数据的结果。尽管 RBM 很有前途，但一定不是 RBM 的终极。

表 7.1　使用 12 种抗原的二分数据的结果

模　　型	敏　感　度	特异性
Logistic 回归	0.697	0.811
RBM	0.720	0.800

敏感度（Sensitivity）是分类器识别积极结果的一种能力，特异性（Specificity）则是区分消极结果的一种能力。

$$Sensitivity = \frac{NTP}{NTP + NTN} \times 100\%$$

$$Specificity = \frac{NTN}{NTP + NTN} \times 100\%$$

式中，NTP 是真阳性的数量；NTN 是阴性的数量。

7.4.2 麻醉镇定作用预测的 RBM

研究人员开发了一个 RBM 来预测麻醉的镇定作用。两种不同类型的特征被提取出来。

第一组特征是与患者的自主神经系统相关的属性，管理着内脏器官的功能，包括心率、血压和外围毛细管氧饱和度（动脉血氧饱和度）。

第二组特征的测量指标与患者的麻醉诱导状态有关，包括局部麻醉浓度、呼气二氧化碳浓度、小部分吸气二氧化碳和最小肺泡浓度。

对 27 名麻醉患者收集包括镇定的响应变量在内的连续数据。训练集由 5000 个随机选择的阶段组成，其中 1000 个用于抽样。用留一法交叉验证均方误差。

按照 benchmark 性能，研究人员首先对 RBM 的均方差与前馈神经网络（ANN）和使用不同特征（MN－1、MN－2、MN－3）的模块化神经网络进行比较。图 7.3 给出了对比结果。表明前馈神经网络、MN－1、MN－2、MN－3 和 RBM 的平均均方差。均方差分别为 0.0665、0.0590、0.0551、0.0515 和 0.0504。RBM 的错误最小。这是使用 RBM 的一个积极的信号，虽然不是决定性的，但均方差似乎和 MN－3 有所区别。

应用该方法到生物系统能够减少对病人的监测成本，并提高医疗质量，其结果令数据科学界震惊。

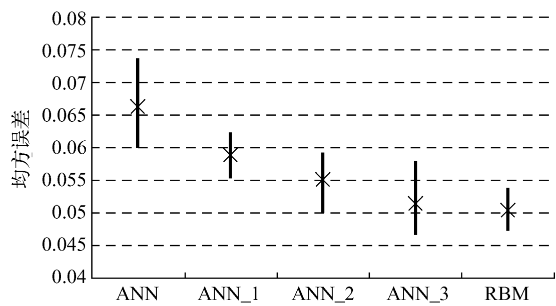


图 7.3　不同建模方法的均方差对比

7.5 RBM 的 R 实现

【例 7.1】用 Mobility 数据结构构建 RBM。

(1) 加载所需的包和数据

```
> require(RcppDL)
> require("ltm")
> data(Mobility)
> data <= Mobility
```

RcppDL 包包含有评估 RBM 的函数，ltm 包含有 Mobility 数据集。采样 1000 个观测值，其中 800 用作训练集，数据结构见表 6.1。

```
> set.seed (2395)
> n = nrow(data)
> sample <- sample (1:n,1000,FALSE)
> data  <- as.matrix (Mobility [sample,])
> n = nrow(data)
> train <- sample (1:n,800,FALSE)
```

(2) 数据准备

```
> x_train  <- matrix(as.  numeric (unlist (data[
train,])),nrow = nrow(data[train,]))
> x_test  <- matrix(as.  numeric (unlist (data[
-train,])),nrow = nrow(data[ -train,]))
> x_train  <- x_train [, -c(4,6)]            #删除训练集第 4 和第 6 列
> x_test  <- x_test [, -c(4,6)]              #删除测试集第 4 和第 6 列
> head (x_ train)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   1    1    1    1    0    1
[2,]   1    0    1    0    0    0
[3,]   1    1    1    0    0    0
[4,]   1    0    0    0    0    0
[5,]   1    0    0    0    0    0
[6,]   1    1    1    0    0    0
```

```
> head (x_test)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    0    0    0
[2,]    1    0    0    0    0    0
[3,]    1    0    0    0    0    0
[4,]    1    0    1    0    0    0
[5,]    1    1    1    1    0    0
[6,]    0    0    1    0    0    0
```

(3) 构建 RBM

```
> fit <- Rrbm (x_train)
```

设定隐含层3个结点的学习率为0.01。

```
> setHiddenRepresentation (fit, x = 3)
> setLearningRate (fit, x = 0.01)
```

模型的摘要：

```
> summary (fit)
$LearningRate
[1] 0.01
$ContrastiveDivergenceStep
[1] 1
$TrainingEpochs
[1] 1000
$HiddenRepresentation
[1] 3
```

模型训练代码：

```
> train (fit)
```

可以在 Rrbm 中设定很多参数，包括：

- setStep signature
- setLearningRate
- setTrainingEpochs

（4）模型部署

```
reconProb <- reconstruct (fit,x_ train)
> head(reconProb,6)
```

```
         [,1]       [,2]       [,3]       [,4]         [,5]         [,6]
[1,]  0.7598778  0.3435336  0.6941071  0.11650449  0.10796628  0.14369203
[2,]  0.8237427  0.3128176  0.7621758  0.06838804  0.06258921  0.08983742
[3,]  0.8187479  0.3163145  0.7565158  0.07186695  0.06600808  0.09402979
[4,]  0.8092837  0.3198017  0.7461960  0.07815092  0.07162580  0.10108005
[5,]  0.8092837  0.3198017  0.7461960  0.07815092  0.07162580  0.10108005
[6,]  0.8187479  0.3163145  0.7565158  0.07186695  0.06600808  0.09402979
```

将概率转化为二进制值：

```
> recon <- ifelse(reconProb > =0.5,1,0)
```

查看重建的值：

```
> head (recon)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   1    0    1    0    0    0
[2,]   1    0    1    0    0    0
[3,]   1    0    1    0    0    0
[4,]   1    0    1    0    0    0
[5,]   1    0    1    0    0    0
[6,]   1    0    1    0    0    0
```

创建混合矩阵：

```
> table(recon,x_train,dnn = c("Predicted","Observed"))
```

```
          Observed
Predicted    0     1
    0      2786   414
    1       371  1229
```

如图 7.4 所示，图像重建虽然不是 RBM 的核心，但包含了 RBM 的主要特征。

```
> par(mfrow = c(1,2))
> image(x_train,main = " Train ")
> image(recon,main = " Reconstruction ")
```

也可以使用 deepnet 包评估 RBM，方法如下：

```
> fit2 <- rbm.train(x_train,
    hidden =3,
    numepochs =3,
    batchsize =100,
    learningrate =0.8,
    learningrate_scale =1,
    momentum =0.5,
    visible_type ="bin",
    hidden_type ="bin",
    cd =1)
```

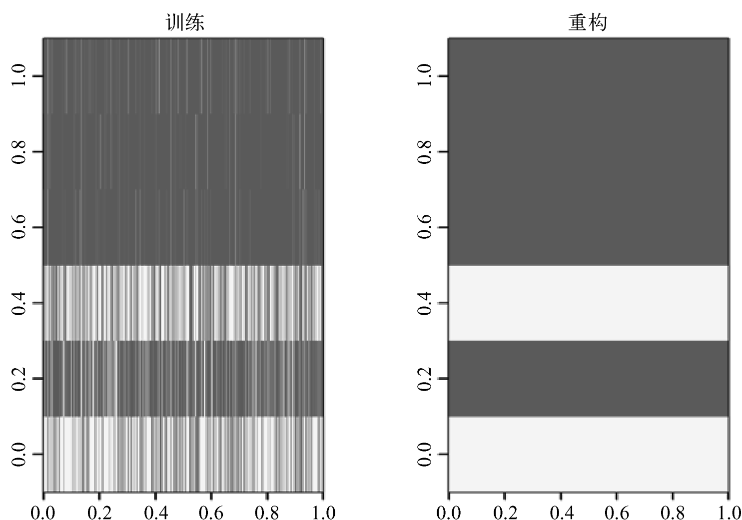


图 7.4　使用 RBM 对 Mobility 的重建

注意，rbm.train 允许指定块的大小，块的大小在大的样本中非常重要。RBM 是非常强大的学习机器，但是它们最重要的用途是作为深度置信网络的构建而使用的学习方法，细节见第 8 章。

7.6　学习指南

RBM 是由一个隐藏层和一个可见层组成的。与前馈网络不同的是，RBM 可见层和隐藏层之间的连接是无向的（两个方向传播），并且是完全连接的（从给定层的每

个单元连接到下一个层的每个单元)，前提是如果允许任何层的任何单元能够连接到任何层，那么就有一个玻耳兹曼机，但是不是 RBM。

标准的 RBM 有一个二进制隐藏层和可见层，也就是，神经元的激活值在伯努利分布下是 0 或 1，没有非线性变换。

虽然 RBM 出现已经有一段时间了，最近推出的对比散度无监督的训练算法重新燃起人们的兴趣。

RBM 是一个可生成的随机神经网络，它能学习对其一组输入的概率分布。

第 8 章　深度置信网络 DBN

一个深度置信网络（Deep Belief Network，DBN）是由多个堆栈 RBM 构成的概率生成多层神经网络。在 DBN 中，每两个相邻隐层构成一个 RBM，RBM 的输出是输入的特征，每一个 RBM 实际上是一个非线性变换，RBM 的输出是下一个 RBM 的输入。堆栈中的每一个 RBM 起到了对数据不同表达的作用。

DBN 一般分为两类层次：一个是可视层；一个是隐层。可视层包括输入结点和输出结点，其他结点为隐藏层结点，DBN 的训练分为两步：预训练和不同于反向传播的微调算法。

8.1　DBN 原理

使用 BP 算法单独训练每一层的时候，必须丢掉网络的第三层，才能级联自联想神经网络。然而，有一种更好的神经网络模型，这就是受限玻耳兹曼机。使用层叠玻耳兹曼机组成深度神经网络的方法，在深度学习里被称作深度置信网络 DBN，这是目前非常流行的方法。下面的术语，将自联想网络称作自编码（Auto Encoder，AE）网络。通过堆栈自编码网络形成 DBN。

经典的 DBN 网络结构是由若干层 RBM 和一层 BP 组成的一种深层神经网络，结构如图 8.1 所示。

DBN 在训练模型的过程中主要分为两步：

(1) 预训练阶段

预训练采用逐层贪婪学习策略，独立地使用对比散度训练每个 RBM，然后堆积在一起。

当第一个 RBM 训练好后，其参数就确定了，通过训练样本（可视层）训练的隐藏层结点的值（RBM 的输出）变成第二个 RBM 可视结点的值。第一个 RBM 学习到的权重用作预测隐层结点的激活概率。激活概率就是第二个 RBM 可视层的权重。重

复这个过程，直到最后 RBM 训练完成。

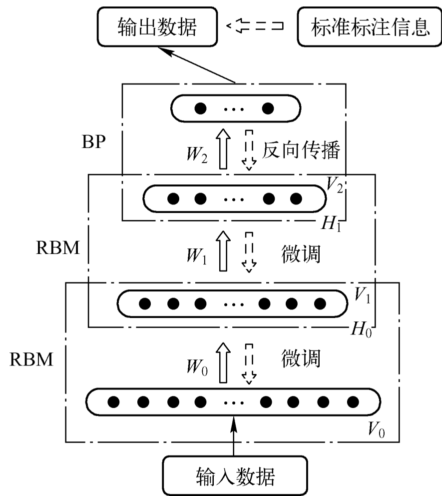


图 8.1　DBN 基本结构

总的来说，与多层神经网络一样，可以通过一层的激活概率作为下一层的训练数据来有效地训练 DBN。

因为预训练是无监督的，所以，这个过程不需要类标签。实际上，分块（Batch－Learning）方法通常被用来加速预训练过程，用 RBM 权重修改每一个块。多个隐藏层产生的多个特征层变得越来越复杂，预训练易于捕获数据的高层特征。在微调阶段，当给定标签后，高层特征有助于有监督学习。

（2）微调阶段

微调阶段采用标准的 BP 算法，自上而下调整预训练网络每一层的权重，使权重更适合分类。正如前面看到的那样，微调包括调整权重，调整权重的目的在于通过减少网络预测类与实际类的误差来增强鉴别能力，另外，Softmax 输出层使用多类分类器。所以，Softmax 分类器通过 RBM 学到了特征抽取联合模型，输出层结点数量等于类别数量。

因为微调包含了有监督学习，所以，训练数据需要相应标签。训练完成后，通过对测试数据从第一个可视层前向传播到 Softmax 输出层来预测测试样本类标签。

8.2　应用案例

由于 AE 可以对原始数据在不同概念的粒度上进行抽象，一种自然的深度网络的

应用是对数据进行压缩（或者叫降维）。

（1）降维

胡邵华等用一种自编码网络实现了对经典的“瑞士卷”数据的重构(见图8.2)。

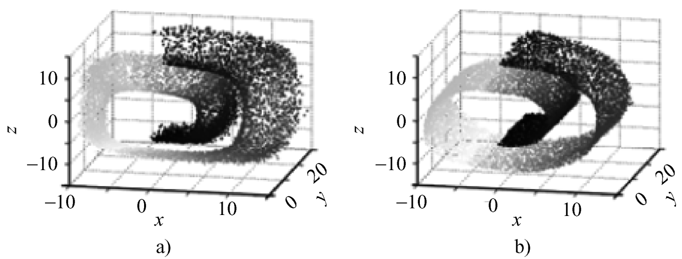


图8.2 “瑞士卷”重构

“瑞士卷”数据是经典的机器学习中难于分类的数据之一，其隐含的数据内在模式难以在二维数据中描述。然而，胡邵华等采用深度置信网络实现了对三维瑞士卷数据的2维表示，其自编码网络结点大小依次为3－100－50－25－10－2。具体的实现细节参考有关文献。

（2）特征提取

通过训练一个5层的深度网络提取音乐的特征，用于音乐风格的分类，其分类精度比基于梅尔倒谱系数（MFCC）特征分类的方法提高了14个百分点（见图8.3）。

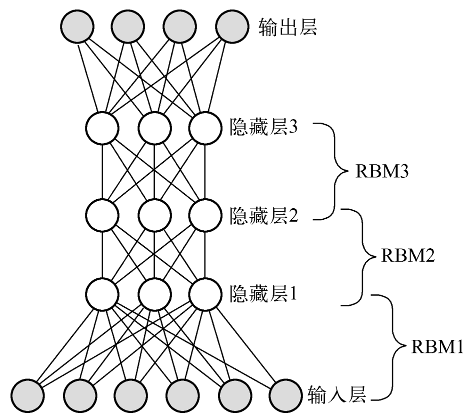


图8.3 深度网络提取音乐的特征

实现思路非常简单，用上述层叠的多个RBM网络组成深度网络结构来提取音乐的特征。输入的原始数据是经过分帧、加窗之后的信号的频谱。分类器采用的是支持向量机（SVM）。对比的方法则是提取MFCC特征系数，分类器同样采用SVM。更

多的细节和实验结果可以参考有关文献。

深度网络是一种良好的机器学习方法，其特征提取功能能够针对不同概念的粒度大小，在很多领域都得到广泛的应用。通常，DBN 对一维数据的建模比较有效，例如语音。而通过级联多层卷积网络组成深度网络的模型主要用于二维数据建模，例如图像等。

8.3　DBN 的 R 实现

【例 8.1】以 Mobility 数据集第 1、2、3、5、7、8 列属性预测某人是否去看电影或看文化展览或参加亲子活动或去母婴馆。

（1）加载依赖的包和数据

```
> gc(rm(list = ls()))                              #将内存变量清空
> library(RcppDL)
> library(ltm)
> data(Mobility)
> data <- Mobility                                 #获取数据
> y <- apply(cbind(data[,4],data[,6]),1,max,na.rm = TRUE)
```

（2）数据准备

```
> set.seed(2)
> n = nrow(data)                              #样本下标随机取样的上界
> sample  <- sample(1:n,1000,FALSE) #随机产生 1000 个样本的下标
> data  <- as.matrix(Mobility[sample,])#根据下标取出 1000 个样本
> n = nrow(data)                              #训练样本下标随机取样的上界
#随机产生 800 个训练样本的下标
> train <- as.integer(sample(row.names(data),800,FALSE))
#格式化训练集：
> y_train <- as.numeric(y[train])         #根据训练样本下标产生 800 个样本标记
> temp  <- ifelse(y_train  ==0,1,0)       #取反
> y_train  <- cbind(y_train,temp)
> head(y_train)                               #查看训练集
```

```
      y_train  temp
[1,]     1       0
[2,]     0       1
[3,]     1       0
[4,]     0       1
[5,]     0       1
[6,]     1       0
```

```
#格式化测试集:
> n1 <- setdiff(sample,train)          #求 sample 的补集
> y_test  <- as.numeric(y[n1])         #根据测试样本下标产生 200 个测试样本标记
> temp1  <- ifelse(y_test ==0,1,0)
> y_test  <- cbind(y_test,temp1)
> head(y_test)                         #查看测试集
```

```
      y_test  temp1
[1,]     0       1
[2,]     0       1
[3,]     1       0
[4,]     0       1
[5,]     1       0
[6,]     0       1
```

检查训练集和测试集的样本数:

```
> nrow(y_ train)                       #显示训练集样本数
[1] 800
> nrow(y_test)                         #显示测试集样本数
[1] 200
```

```
#下面创建训练集和测试集对象(x_train,x_test)
> data <- as.data.frame(data)
> x_train <- as.matrix(data[as.character(train),])
> x_test <- as.matrix(data[as.character(n1),])
> x_train  <- x_train [,-c(4,6)]                 #删除第 4、6 列
#把 int 类型矩阵转换为 num 类型
> x_train <- as.data.frame(x_train)
```

```
>x_train <- lapply(x_train,as.numeric)
>x_train <- as.data.frame(x_train)
>x_train <- as.matrix(x_train)
>x_test  <- x_test [, -c(4,6)]              #删除第 4、6 列
>x_test <- as.data.frame(x_test)
>x_test <- lapply(x_test,as.numeric)
>x_test <- as.data.frame(x_test)
>x_test <- as.matrix(x_test)
>head (x_train)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    1    1    0    1
[2,]    1    0    1    0    0    0
[3,]    1    1    1    0    0    0
[4,]    1    0    0    0    0    0
[5,]    1    0    0    0    0    0
[6,]    1    1    1    0    0    0
```

```
>head (x_test)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    0    0    0
[2,]    1    0    0    0    0    0
[3,]    1    0    0    0    0    0
[4,]    1    0    1    0    0    0
[5,]    1    1    1    1    0    0
[6,]    0    0    1    0    0    0
```

(3）建模

创建两个隐藏层的网络，第一个隐藏层为 12 个结点，第二个隐藏层为 10 个结点。

```
>hidden = c(12,10)
>fit  <- Rdbn(x_train,y_train,hidden)       #创建具有两个隐藏层的 RBN 模型
>summary(fit)                               #查看模型细节
```

```
  $PretrainLearningRate
[1] 0.1
  $PretrainingEpochs
```

```
[1] 1000
    $FinetuneLearningRate
[1] 0.1
    $FinetuneEpochs
[1] 500
    $ContrastiveDivergenceStep
[1] 1
```

（4）模型部署

```
> pretrain (fit)                                    #预训练
> finetune (fit)                                    #微调
> predProb <- predict (fit,x_test)                  #用 RBN 模型预测测试样本分类
> head (predProb,6)                                 #输出是预测概率
```

```
          [,1]       [,2]
[1,]  0.3942584  0.6057416
[2,]  0.4137407  0.5862593
[3,]  0.4137407  0.5862593
[4,]  0.4332217  0.5667783
[5,]  0.4970219  0.5029781
[6,]  0.4142550  0.5857450
```

```
> pred1 <- ifelse(predProb[,1] >= 0.5,1,0)                         #预测概率二值化
> table(pred1,y_test[,2],dnn = c("Predicted","Observed")) #计算混淆矩阵
```

```
          Observed
Predicted   0   1
    0     115  75
    1       0  10
```

8.4 学习指南

DNN 有两种基本结构：卷积神经网络（Convolutional Neural Networks，CNN）和深度置信网络（Deep Belief Network，DBN）。

从前几章已经发现，AE 隐藏层和 RBM 能作为有效的特征探测器，但是很少直接使用它们。实际数据比前面的规则数据集更加复杂，所以，需要寻找到一些方法

间接使用这个特征检测器。

幸运的是，发现这些 RBM 可以堆叠（Stack）形成深度网络，这个网络可以使用经典的反向传播算法，逐层贪婪训练，以帮助克服梯度弥失和过度拟合问题（见图 8.4）。

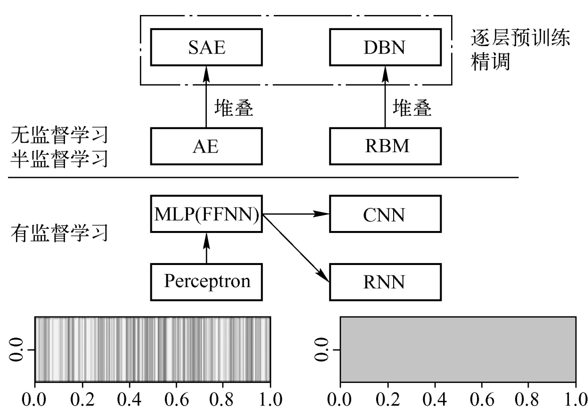


图 8.4　DBN 结构

DBN 的灵活性使得它的拓展比较容易。一个拓展就是加入卷积的 DBN（Convolutional Deep Belief Networks，CDBN）。DBN 并没有考虑到图像的二维结构信息，因为输入是简单地从一个图像矩阵一维向量化的。而 CDBN 就是考虑到了这个问题，它利用邻域像素的空域关系，通过一个称为卷积 RBM 的模型来达到生成模型的变换不变性，而且可以容易地变换到高维图像。DBN 并没有明确地处理观测变量为时间序列的情况，虽然目前已经有这方面的研究，例如堆叠时间 RBM，时态卷积网络。这种序列学习的应用，给语音信号处理问题带来了未来的研究方向。

第 9 章　MXNetR

9.1　MXNet 技术特性

MXNet 名字源于“Mix and Maximize”，与其他工具相比，MXNet 结合了符号语言和过程语言的编程模型，并试图最大化各自优势，利用统一的执行引擎进行自动多 GPU 并行调度优化。

不同的编程模型有各自的优势，以往的深度学习库往往着重于灵活性，或者性能。MXNet 通过融合的方式把各种编程模型整合在一起，并且通过统一的轻量级运行引擎进行执行调度。这使得用户可以直接复用稳定高效的神经网络模块，并且可以通过 R 等高级语言进行快速扩展。

MXNet 融合了 Minerva 的动态执行、Cxxnet 的静态优化和 Purine2 的符号计算等思想，直接支持 C 接口和静态/动态 Library，使得对于新语言的扩展更加容易。

MXNet 主要技术特点如下：

1）轻量级调度引擎。在数据流调度的基础上引入了读写操作调度，并且使得调度和调度对象无关，用以直接有机支持动态计算和静态计算的统一多 GPU 多线程调度，使得上层实现更加简洁灵活。

2）符号计算支持。MXNet 支持基于静态计算流图符号计算。计算流图不仅使设计复杂网络更加简单快捷，而且基于计算流图，MXNet 可以更加高效地利用内存。同时进一步优化了静态执行的规划，内存需求比 Cxxnet 还要少。

3）混合执行引擎。相比 Cxxnet 的全静态执行、Minerva 的全动态执行，MXNet 采用动静态混合执行引擎，可以把 Cxxnet 静态优化的效率和 Ndarray 动态运行的灵活性结合起来，把高效的 C ++ 库更加灵活地和 Python 等高级语言结合在一起。

4）更加灵活。在 MShadow C ++ 表达式模板的基础上，符号计算和 NDArray 使在 R 等高级语言内编写优化算法、损失函数和其他深度学习组件并高效无缝地支持

CPU/GPU 成为可能。用户无须关心底层实现，在符号和 NDArray 层面完成逻辑即可进行高效的模型训练和预测。

5）对于云计算更加友好。所有数据模型可以从 S3/HDFS/Azure 上直接加载训练。

6）代码更加简洁高效。大量使用 C ++11 特性，使 MXNet 利用最少的代码实现尽可能最大的功能。用约 11k 行 C ++代码（加上注释 4k 行）实现了以上核心功能。

7）开源用户和设计文档。MXNet 提供了非常详细的用户文档和设计文档以及样例。所有的代码都有详细的文档注释，并且会持续更新代码和系统设计细节，对广大深度学习系统开发和爱好者有所帮助。

8）它能够在 R 上编写无缝的多核 GPU 张量/矩阵运算。

9）它能够在 R 上构建和定制最先进水平的深度学习模型，并将它们应用到图像分类和数据科学等挑战性任务。

9.2 MXNetR 安装

MXNetR 是最大开源分布式机器学习项目 DMLC 发布的深度学习框架 MXNet 的一个 R 包，它填补了 R 语言在深度学习和 GPU 计算上的空白，可以直接使用 R 进行显卡加速深度神经网络训练。MXNetR 支持 Amazon Linux、Ubuntu/Debian、OS X 和 Windows，并且支持多语言编程。

9.2.1 安装 MXNet 基本需求

（1）最小需求

C ++11 编译器。用于编译 C ++ 和构建 MXNet 源代码，MXNet 支持的编译器包括：

1）G ++ 。C ++4.8 以后版本。

2）Clang。Clang 是一个 C ++ 编写、基于 LLVM、发布于 LLVM BSD 许可证下的 C/C ++/Objective – C/Objective – C ++ 编译器。它与 GNU C 语言规范几乎完全兼容（当然，也有部分不兼容的内容，包括编译命令选项也会有点差异），并在此基础上增加了额外的语法特性，比如 C 函数重载（通过_attribute_（（overloadable））来修饰函数），其目标（之一）就是超越 GCC。

3）BLAS（Basic Linear Algebra Subprograms）库。BLAS 即基础线性代数子程序

库，里面拥有大量已经编写好的关于线性代数运算的程序，支持 BLAS 的库包括 Lib-blas、Openblas 和 Intel MKL。

(2) 使用 GPU 的需求

支持计算能力大于 Compute Capability 2.0 或更高的 GPU，GPU 的特性取决于 CUDA，最好使用 CUDA Toolkit 7.0 或更高版本。CUDA Toolkit 是一个允许 MXNet 在 NVIDIA GPU 上运行的环境，包括编译器、数学库和调试工具。

CuDNN (CUDA Deep Neural Network) 库，通过协调 CPU 性能来加速 GPU。

(3) 支持计算机视觉和图像增强的需求

如果要处理计算机视觉方面的任务，则需要 OpenCV 库。

9.2.2 MXNet 云设置

如果要使用 MXNet 云，则需求如下：

(1) 通过 AWS 预配置 Amazon Machine Images (AMI)

下面的链接给出了云上深度学习 Blog 和 AMI 的示例：

https://aws.amazon.com/cn/blogs/aws/new-p2-instance-type-for-amazon-ec2-up-to-16-gpus/ (Deep Learning Blog)

https://aws.amazon.com/marketplace/pp/B01M0AXXQB (Deep Learning Blog)

其中，AMI 不但支持 MXNet，还支持其他深度学习框架。

(2) 在 AWS 上使用 MXNet 的示例

为了扩展在 AWS GPU 上使用 CloudFormation 模板的示例，需要阅读以下链接：

https://aws.amazon.com/cn/blogs/compute/distributed-deep-learning-made-easy/ (CloudFormation Template AWS Blog)

9.2.3 MXNet 安装方法

(1) MXNet 快速安装

如果安装环境是 Amazon Linux 或 Ubuntu，则可以使用 Git Bash 脚本快速安装 MXNet。

(2) MXNet 标准安装

这里只给出 Windows 的安装，Linux 和 OS X (Mac) 安装可参考相关文献。

(3) 下载 MXNet Prebuilt 包

Prebuilt 包包括 MXNet 库和所有第三方依赖库。Prebuilt 包有两个版本，一个是

支持 GPU，一个是不支持 GPU。把 Prebuilt 包解压到文件夹，并起一个合适的名字，如 D:\MXNet，打开该文件夹，执行 setupenv. cmd，最后用一个 C ++ 例子测试，产生的库称为 libmxnet. dll。

（4）Building and Installing Packages on Windows

为了安装 MXNet 需要下面的依赖软件：

- Microsoft Visual Studio 2013；
- Visual C ++ Compiler Nov 2013 CTP（提供 C ++ 支持）；
- OpenCV（支持计算机视觉操作）；
- OpenBlas（支持线性戴氏操作）；
- CuDNN（提供 Deep Neural Network 库）。

（5）buildMXNet 源代码

- 从 GitHub 下载 MXNet 源代码；
- 用 CMake 在/build 创建 Visual Studio 解决方案；
- 打开 . sln 并编译，产生的库称为 mxnet. dll，存放在“. /build/Release/”或“. /build/Debug”文件夹内。

9.2.4 MXNetR 安装方法

（1）不使用 GPU 的安装

方案 1：使用 prebuilt binary 包。

对于 Windows 和 OS X（Mac）用户，MXNet 提供了 prebuilt binary 包。该包每周更新，在 R 控制台安装 prebuilt binary 包命令：

```
install.packages("drat",repos = "https://cran.rstudio.com")
drat:::addRepo("dmlc")
install.packages("mxnet")
```

方案 2：根据源代码创建 Library。

步骤 1. 创建共享 Library：根据 MXNet C ++ 源代码创建共享 Library。

步骤 2. 执行下面命令安装 MXNet 依赖软件并创建 MXNetR 包：

```
Rscript -e "install.packages('devtools',repo = 'https://cran.rstudio.com')"
cd R-package
Rscript -e "library(devtools); library(methods);
options(repos = c(CRAN = 'https://cran.rstudio.com')); install_deps(dependencies =
```

```
TRUE)"
cd ..
makerpkg
```

注：R - package 在 MXNet 中是个文件夹。

下面的命令创建 MXNetR 包为“tar. gz”文件：

```
R CMD INSTALLmxnet_0.7.tar.gz
```

（2）使用 GPU 的安装

为了使用 GPU 安装 MXNet，需求如下：

- Microsoft Visual Studio 2013；
- NVidia CUDA Toolkit；
- MXNet package；
- CuDNN（提供 Deep Neural Network 库）。

从 https://github.com/dmlc/mxnet/下载的 MXNet 包是“*.zip”文件，需要解压到"/mxnet/R - package"文件夹内。

从 https://github.com/dmlc/mxnet/releases 下载最新的 GPU - enabled MXNet 包，解压到“/nocudnn”文件夹内。

下载并安装 CuDNN V3。网络获取下载链接，需要注册为 NVIDIA 用户，解压后将看到三个文件夹“/bin”“/include”和“/lib”，复制到“nocudnn/3rdparty/cudnn/”，也可以直接解压 *.zip 文件到“/nocudnn”。

创建文件夹 R - package/inst/libs/x64。MXNet 只支持 64 位操作系统，复制以下（.dll）文件到 R - package/inst/libs/x64 内（共 11 个）：

1）nocudnn/lib/libmxnet.dll。

2）文件夹 nocudnn/3rdparty/directory 下的 *.dll。

3）文件夹/bin 下的 cudnn.dll 和 openblas.dll。

复制“nocudnn/include/”下文件到 R - package/inst/。这时有了 R - package/inst/include/等三个子文件夹。

确保 R 的安装路径已在环境变量 PATH 中。

执行 R CMD INSTALL——no - multiarch R - package。

注：MXNet 库是用 Rcpp 建立的。

9.2.5 常见的安装问题

（1）Mac OS X 错误信息

信息：link error ld：没有找到 lgomp 库。

原因：OpenMP 的 GNU 实例路径不在系统的 library 路径。

解决：在系统 library 路径里添加 OpenMP library 路径。

通过执行下面命令创建局部数据库：

```
sudo launchctl load -w /System/Library/LaunchDaemons/com.apple.locate.plist
```

确定 OpenMP library 路径：locate libgomp.dylib。

为了把 OpenMP library 路径添加到系统 library 路径中，用上面的输出替换下面命令中的“path1”：

```
ln -s path1 /usr/local/lib/libgomp.dylib
```

执行下面命令完成路径的添加：

```
make -j$(sysctl -n hw.ncpu)
```

（2）R 错误信息

信息：CUDA 不能加载 MXNet。

解决：如果在安装 MXNet 时，已经是 CUDA 可用，仍不能加载 MXNet，把下面的命令添加到环境变量$RHOME/etc/ldpaths 中。

```
export CUDA_HOME=/usr/local/cuda
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64:${LD_LIBRARY_PATH}
```

注：在 R 中，用 R.home( )查看环境变量$RHOME。

9.3 MXNetR 在深度学习中的应用

9.3.1 二分类模型

【例 9.1】 借用 mxnet 包中的 mx.mlp 函数和 mlbench 包数据设计一个二分类网络。

（1）准备数据，并进行简单的预处理

借用 mlbench 包中的一个二分类数据，并且将它分成训练集和测试集。

```
> require(mlbench)
> require(mxnet)
> data(Sonar,package = "mlbench")
> Sonar[ ,61] = as. numeric(Sonar[ ,61]) - 1
> train. ind = c(1:50,100:150)
> train. x = data. matrix(Sonar[train. ind,1:60])
> train. y = Sonar[train. ind,61]
> test. x = data. matrix(Sonar[ - train. ind,1:60])
> test. y = Sonar[ - train. ind,61]
```

(2) 建模

mxnet 提供了一个训练多层神经网络的函数 mx. mlp，可以通过它来训练一个神经网络模型。

```
> mx. set. seed(0)
> model <- mx. mlp(train. x,                      #训练数据
> train. y,                                       #响应变量
> hidden_node = 10,                               #隐藏层的结点数量
> out_node = 2,                                   #输出层的结点数
> out_activation = "softmax",                     #激活函数
> num. round = 20,
> array. batch. size = 15,
> learning. rate = 0. 07,                         #学习率
> momentum = 0. 9,
> eval. metric = mx. metric. accuracy             #损失函数类型
)
Auto detect layout of input matrix, userowmajor. .
Start training with 1 devices
[1] Train - accuracy = 0. 488888888888889
[2] Train - accuracy = 0. 514285714285714
[3] Train - accuracy = 0. 514285714285714
 ⋮
[18] Train - accuracy = 0. 838095238095238
[19] Train - accuracy = 0. 838095238095238
[20] Train - accuracy = 0. 838095238095238
```

这里要注意，使用 mx. set. seed 而不是 R 自带的 set. seed 函数来控制随机数。因为 mxnet 的训练过程可能会运行在不同的运算硬件上，这里需要一个足够快的随机数生成器来管理整个随机数生成的过程。

（3）模型部署

模型训练好之后，可以很简单地进行预测：

```
> preds = predict( model, test. x)
Auto detect layout of input matrix, userowmajor. .
> pred. label = max. col( t( preds) ) - 1
> table( pred. label, test. y)
            test. y
 pred. label   0    1
      0       24   14
      1       36   33
```

如果进行的是多分类预测，mxnet 的输出格式是类数 × 样本数。

9. 3. 2　回归模型与自定义神经网络

mx. mlp 接口固然很方便，但是神经网络的一大特点便是它的灵活性，不同的结构可能有着完全不同的特性。mxnet 的亮点之一便是它赋予了用户极大的自由度，从而可以任意定义需要的神经网络结构。本节用一个简单的回归任务来介绍其相关的语法。

【例 9. 2】使用 mxnet 提供了“Symbol”系统，建立回归预测模型。

（1）数据准备

```
> data( BostonHousing, package = " mlbench" )
> train. ind = seq( 1, 506, 3)
> train. x = data. matrix( BostonHousing[ train. ind, - 14] )
> train. y = BostonHousing[ train. ind, 14]
> test. x = data. matrix( BostonHousing[ - train. ind, - 14] )
> test. y = BostonHousing[ - train. ind, 14]
```

（2）建模

mxnet 提供了一个叫作“Symbol”的系统，从而可以定义结点之间的连接方式与激活函数等参数。下面是一个定义没有隐藏层神经网络的简单例子：

```
> data <- mx. symbol. Variable( "data" )                # 定义输入数据
> fc1 <- mx. symbol. FullyConnected( data, num_hidden = 1 )
> lro <- mx. symbol. LinearRegressionOutput( fc1 )      #定义损失函数
```

在神经网络中，回归与分类的差别主要在于输出层的损失函数。这里使用平方误差来训练模型。希望进一步了解 Symbol 的读者可以继续阅读相关文档。

定义了神经网络之后，便可以使用 mx. model. FeedForward. create 进行训练了。

```
> mx. set. seed(0)
> model <- mx. model. FeedForward. create( lro,
    X = train. x,
    y = train. y,
    ctx = mx. cpu( ),
    num. round = 50,
    array. batch. size = 20,
    learning. rate = 2e - 6,
    momentum = 0. 9,
    eval. metric = mx. metric. rmse)
Auto detect layout of input matrix, userowmajor. .
Start training with 1 devices
[1] Train - rmse = 16. 063282524034
[2] Train - rmse = 12. 2792375712573
[3] Train - rmse = 11. 1984634005885
⋮
[48] Train - rmse = 8. 26890902770415
[49] Train - rmse = 8. 25728089053853
[50] Train - rmse = 8. 24580511500735
```

（3）模型部署

这里针对回归任务修改了 eval. metric 参数。模型提供的评价函数包括“accuracy”“rmse”“mae” 和 “rmsle”，用户也可以针对需要自定义评价函数，例如：

```
> demo. metric. mae <- mx. metric. custom( "mae", function( label, pred) {
  res <- mean( abs( label - pred) )
  return( res)
})
> mx. set. seed(0)
```

```
> model <- mx.model.FeedForward.create(lro,
            X = train.x,
            y = train.y,
            ctx = mx.cpu(),
            num.round = 50,
            array.batch.size = 20,
            learning.rate = 2e-6,
            momentum = 0.9,
            eval.metric = demo.metric.mae)
```

```
Auto detect layout of input matrix,userowmajor..
Start training with 1 devices
[1] Train-mae=13.1889538083225
[2] Train-mae=9.81431959337658
[3] Train-mae=9.21576419870059
⋮
[48] Train-mae=6.41731406417158
[49] Train-mae=6.41011292926139
[50] Train-mae=6.40312503493494
```

至此，mxnet 使用方法已经基本掌握了。9.3.3 节将介绍更有趣的应用。

9.3.3 手写数字竞赛

【例 9.3】 以 Kaggle 上的手写数字数据集（MNIST）竞赛为例子，通过 mxnet 定义卷积神经网络，并在 GPU 上快速训练模型。

（1）数据准备

从 Kaggle 上下载数据，并将它们放入“data/”文件夹中。然后读入数据，并做一些预处理工作。

```
> require(mxnet)
> train <- read.csv('data/train.csv', header = TRUE)
> test <- read.csv('data/test.csv', header = TRUE)
> train <- data.matrix(train)
> test <- data.matrix(test)
```

```
> train. x <- train[ , -1]
> train. y <- train[ ,1]
> train. x <- t(train. x/255)
> test <- t(test/255)
```

最后两行预处理的作用有两个：

1）原始灰度图片数值处在[0,255]之间，将其变换到[0,1]之间。

2）mxnet 接受像素 X 图片的输入格式，所以对输入矩阵进行了转置。

（2）建模

LeNet 神经网络结构是由 Yann LeCun 提出的，并用于识别手写数字，也是最早的卷积神经网络之一。同样地，这里使用 Symbol 语法来定义，不过这个结构会比较复杂。

```
> data <- mx. symbol. Variable('data')
#第一个卷积
> conv1 <- mx. symbol. Convolution(data = data, kernel = c(5,5), num_filter = 20)
> tanh1 <- mx. symbol. Activation(data = conv1, act_type = "tanh")
> pool1 <- mx. symbol. Pooling(data = tanh1, pool_type = "max",
                kernel = c(2,2), stride = c(2,2))
#第二个卷积
> conv2 <- mx. symbol. Convolution(data = pool1, kernel = c(5,5),
                num_filter = 50)
> tanh2 <- mx. symbol. Activation(data = conv2, act_type = "tanh")
> pool2 <- mx. symbol. Pooling(data = tanh2, pool_type = "max",
                kernel = c(2,2), stride = c(2,2))
#第一个全连接
> flatten <- mx. symbol. Flatten(data = pool2)
> fc1 <- mx. symbol. FullyConnected(data = flatten, num_hidden = 500)
> tanh3 <- mx. symbol. Activation(data = fc1, act_type = "tanh")
#第二个全连接
> fc2 <- mx. symbol. FullyConnected(data = tanh3, num_hidden = 10)
> lenet <- mx. symbol. SoftmaxOutput(data = fc2)   #损失函数
```

为了让输入数据的格式能对应 LeNet，需要将数据变成 R 中的 array 格式。

```
> train. array <- train. x
> dim(train. array) <- c(28,28,1,ncol(train. x))
> test. array <- test
> dim(test. array) <- c(28,28,1,ncol(test))
```

接下来，分别使用 CPU 和 GPU 来训练这个模型，从而展现不同的训练效率。

```
> n. gpu <- 1
> device. cpu <- mx. cpu()
> device. gpu <- lapply(0:(n. gpu - 1),function(i){ mx. gpu(i)})
```

将 GPU 的每个核以 list 的格式传递进去，如果有 BLAS 等自带矩阵运算并行的库存在，则没必要对 CPU 这么做了。

先在 CPU 上进行训练，只进行一次迭代。

```
> mx. set. seed(0)
> tic <- proc. time()
> model <- mx. model. FeedForward. create(lenet,
        X = train. array,
        y = train. y,
        ctx = device. cpu,
        num. round = 1,
        array. batch. size = 100,
        learning. rate = 0. 05,
        momentum = 0. 9,
        wd = 0. 00001,
        eval. metric = mx. metric. accuracy,
        epoch. end. callback = mx. callback. log. train. metric(100))
```

```
Start training with 1 devices
Batch [100] Train - accuracy = 0. 1066
Batch [200] Train - accuracy = 0. 16495
Batch [300] Train - accuracy = 0. 401766666666667
Batch [400] Train - accuracy = 0. 537675
[1] Train - accuracy = 0. 557136038186157
```

```
print(proc. time() - tic)
```

```
   user    system   elapsed
130.030   204.976   83.821
```

在 CPU 上训练一次迭代一共花了 83 s。接下来在 GPU 上训练 5 次迭代：

```
mx.set.seed(0)
tic <- proc.time()
model <- mx.model.FeedForward.create(lenet,
        X = train.array,
        y = train.y,
        ctx = device.gpu,
        num.round = 5,
        array.batch.size = 100,
        learning.rate = 0.05,
        momentum = 0.9,
        wd = 0.00001,
        eval.metric = mx.metric.accuracy,
        epoch.end.callback = mx.callback.log.train.metric(100))
```

```
Start training with 1 devices
Batch [100] Train-accuracy=0.1066
Batch [200] Train-accuracy=0.1596
Batch [300] Train-accuracy=0.3983
Batch [400] Train-accuracy=0.533975
[1] Train-accuracy=0.553532219570405
Batch [100] Train-accuracy=0.958
Batch [200] Train-accuracy=0.96155
Batch [300] Train-accuracy=0.966100000000001
Batch [400] Train-accuracy=0.968550000000003
[2] Train-accuracy=0.969071428571432
Batch [100] Train-accuracy=0.977
Batch [200] Train-accuracy=0.97715
Batch [300] Train-accuracy=0.979566666666668
Batch [400] Train-accuracy=0.980900000000003
[3] Train-accuracy=0.981309523809527
Batch [100] Train-accuracy=0.9853
```

```
Batch [200] Train - accuracy = 0.985899999999999
Batch [300] Train - accuracy = 0.986966666666668
Batch [400] Train - accuracy = 0.988150000000002
[4] Train - accuracy = 0.988452380952384
Batch [100] Train - accuracy = 0.990199999999999
Batch [200] Train - accuracy = 0.98995
Batch [300] Train - accuracy = 0.990600000000001
Batch [400] Train - accuracy = 0.991325000000002
[5] Train - accuracy = 0.991523809523812
> print(proc.time() - tic)
  user   system  elapsed
 9.288   1.680    6.889
```

在 GPU 上训练 5 轮迭代只花了不到 7 s，快了数十倍！可以看出，对于这样的网络结构，GPU 的加速效果是非常显著的。有了快速训练的办法，可以很快地做预测，并且提交到 Kaggle 上。

```
> preds <- predict(model, test.array)
> pred.label <- max.col(t(preds)) - 1
> submission <- data.frame(ImageId = 1:ncol(test), Label = pred.label)
> write.csv(submission, file = 'submission.csv', row.names = FALSE,
    quote = FALSE)
```

9.3.4 图像识别应用

其实对于神经网络当前的应用场景而言，识别手写数字已经不足为奇。早些时候，Google 公开了一个云 API，让用户能够检测一幅图像里面的内容。本节提供一个例子，让读者能够自制一个图像识别的在线应用。

【例 9.4】 搭建一个 Shiny 应用，实现物体识别。

（1）加载依赖的包和数据

```
install.packages("shiny", repos = "https://cran.rstudio.com")
install.packages("imager", repos = "https://cran.rstudio.com")
```

现在已经配置好了 mxnet、shiny 和 imager 三个 R 包，下一步则是让 Shiny 直接下载并运行准备好的代码。

```
shiny::runGitHub("thirdwing/mxnet_shiny")
```

第一次运行这个命令会花上几分钟时间下载预先训练好的模型。训练的模型叫Inception－BatchNorm Network，如果读者对它感兴趣，可以阅读相关文献。准备就绪之后，浏览器中会出现一个网页应用，就能用本地或在线图片来测试。

（2）导入预训练过的模型文件

```
> model <- mx.model.load("Inception/Inception_BN", iteration = 39)
> synsets <- readLines("Inception/synset.txt")
> mean.img <- as.array(mx.nd.load("Inception/mean_224.nd")[["mean_img"]])
```

（3）图像预处理

使用一个自定义函数对图像进行预处理，这个步骤对于神经网络模型而言至关重要。

```
preproc.image <- function(im, mean.image) {
  # crop the image
  shape <- dim(im)
  short.edge <- min(shape[1:2])
  yy <- floor((shape[1] - short.edge)/2) + 1
  yend <- yy + short.edge - 1
  xx <- floor((shape[2] - short.edge)/2) + 1
  xend <- xx + short.edge - 1
  croped <- im[yy:yend, xx:xend, , ]
  # resize to 224 x 224, needed by input of the model.
  resized <- resize(croped, 224, 224)
  # convert to array(x, y, channel)
  arr <- as.array(resized)
  dim(arr) = c(224, 224, 3)
  #substract the mean
  normed <- arr - mean.img
  # Reshape to format needed bymxnet(width, height, channel, num)
  dim(normed) <- c(224, 224, 3, 1)
  return(normed)
}
```

（4）模型部署

读入图像，预处理与预测代码如下：

```
im <- load.image(src)
normed <- preproc.image(im,mean.img)
prob <- predict(model,X = normed)
max.idx <- order(prob[,1],decreasing = TRUE)[1:5]
result <- synsets[max.idx]
```

9.4 学习指南

MXNet 是一个在底层与接口都有着丰富功能的软件，如果读者对它感兴趣，可以参考一些额外的材料来进一步了解 MXNet。

第 10 章　word2vec 的 R 语言实现

10.1　word2vec 词向量由来

在 word2vec 产生前，还有一些语言模型，起到确定性作用的是词向量。在词向量提出之前还有一些基础性的模型，如统计语言模型、神经网络概率语言模型。

10.1.1　统计语言模型

统计语言模型（也称为 n 元模型，或 n－gram）的一般形式比较直观，一个基本假设是在不改变词语在上下文中的顺序前提下，距离相近的词语关系越近，距离较远的词语关联度越远，当距离足够远时，词语之间则没有关联。

但该模型没有完全利用语料的信息，比如：

1）没有考虑距离更远的词语与当前词的关系，即超出范围 n 的词被忽略了，但这两者很可能是有关系的。

例如，“华盛顿是美国的首都”是当前语句，隔了大于 n 个词的地方又出现了“北京是中国的首都”，在 n 元模型中“华盛顿”和“北京”是没有关系的，然而这两个句子却隐含了语法及语义关系，即“华盛顿”和“北京”都是名词，并且分别是美国和中国的首都。

2）忽略了词语之间的相似性，即统计语言模型无法考虑词语的语法关系。

例如，语料中的“鱼在水中游”应该能够帮助我们产生“马在草原上跑”这样的句子，因为两个句子中“鱼”和“马”、“水”和“草原”、“游”和“跑”、“中”和“上”具有相同的语法特性。

而在神经网络概率语言模型中，充分利用了这两种信息。

10.1.2　神经网络概率语言模型

神经网络概率语言模型（Neural Network Language Model，NNLM）是一种新兴的自然语言处理算法，该模型通过学习训练语料获取词向量和概率密度函数。词向量是多维实数向量，向量中包含了自然语言中的语义和语法关系，词向量之间余弦距离的大小代表了词语之间关系的远近，词向量的加减运算则是计算机在“遣词造句”。

如今在架构方面有比 NNLM 更简单的 CBOW 模型、Skip - gram 模型；其次在训练方面，出现了分层 Softmax 算法、负采样算法，以及为了减小频繁词对结果准确性和训练速度的影响而引入的欠采样（Subsumpling）技术。

图 10.1 给出了基于三层神经网络的自然语言估计模型。它可以计算某一个上下文的下一个词为 w_i的概率，即（$w_i = i \mid context$），词向量是其训练的副产物。NNLM 根据语料库 C 生成对应的词汇表 V。

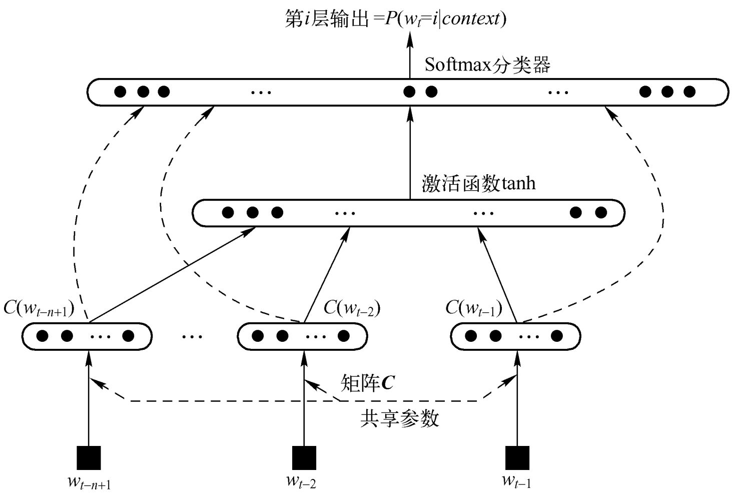


图 10.1　三层 NNLM

10.2　word2vec——词向量特征提取模型

10.2.1　词向量

先解释一下词向量：将词用“词向量”的方式表示可谓是将深度学习算法引入

自然语言处理（NLP）领域的一个核心技术。自然语言理解问题转化为机器学习问题的第一步都是通过一种方法把这些符号数学化。

词向量具有良好的语义特性，是表示词语特征的常用方式。词向量的每一维的值代表一个具有一定的语义和语法上解释的特征，故可以将词向量的每一维称为一个词语特征。词向量是一种词的分布式表示方法，是一种低维实数向量。

例如，NLP 中最直观、最常用的词表示方法是 One - hot。每个词用一个很长的向量（字典的个数）表示，绝大多数是 0，只有一个维度是 1，代表当前词在字典中的位置。例如，“话筒”表示为[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 …]。

但这种 One - hot 表示的是稀疏矩阵，在解决某些任务时会造成维数灾难，而使用低维的词向量就很好地解决了该问题。同时从实践上看，高维的特征如果要套用深度学习算法，其复杂度几乎是难以接受的。

分布式表示的低维实数向量，如[0. 792,0. 177,0. 107,0. 109,0. 542,…]，它让相似或相关的词在距离上更加接近。它的每一维表示词语的一个潜在特征，该特征捕获了有用的句法和语义特征。其特点是将词语的不同句法和语义特征分布到它的每一个维度上来表示。

再来看看结合深度学习算法的词向量是如何获取的。谈到 word2vec 的词向量，就得提到两个模型——CBOW（Continuous Bag - Of - Words，即连续的词袋）以及 Skip - gram 模型。这两个模型都是根据上下文词，来推断当前词发生的概率。可以实现两个基本目标：一是这句话是否是自然语句；二是获得词向量。图 10. 2 示意的是 CBOW 模型结构，图 10. 3 示意的是 Skip - gram 模型结构。

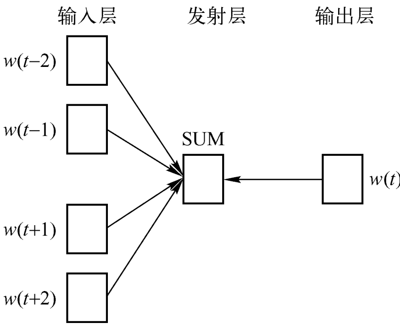


图 10. 2　CBOW 模型结构

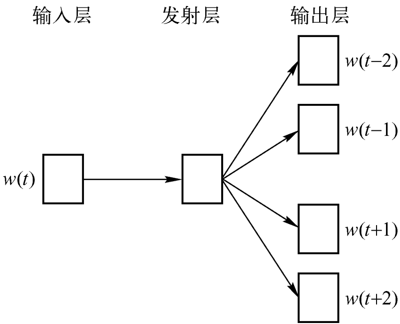


图 10. 3　Skip - gram 模型结构

10. 2. 2　CBOW 的分层网络结构——HCBOW

图 10. 4 示意了 CBOW 的分层网络结构，其中第一层，也就是最上面的那一层称

为输入层。输入的是若干个词的词向量，在神经网络概率语言模型中，从隐含层到输出层的计算量是主要影响训练效率的地方，CBOW 和 Skip - gram 模型考虑去掉隐含层。实践证明，新训练的词向量的精确度可能不如 NNLM 模型（具有隐含层），但可以通过增加训练语料的方法来完善。第三层是方框里面的二叉树，叫霍夫曼树，W 代表一个词，W_{syn1} 代表非叶子结点，是一类词的集合，可以继续分下去。

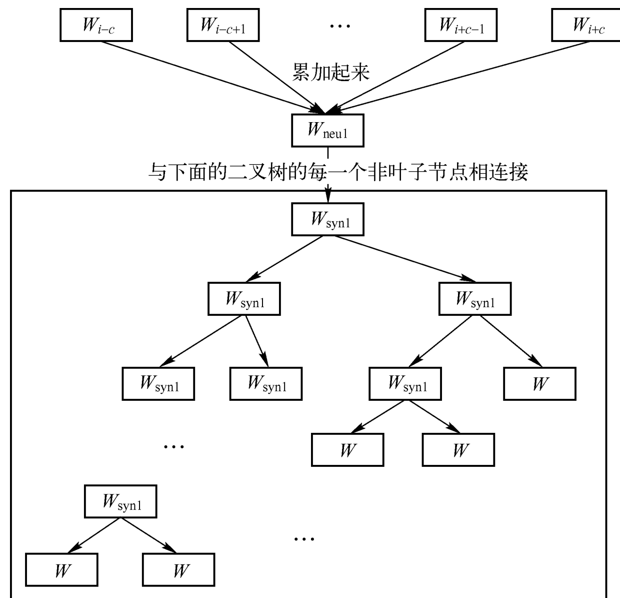


图 10.4　HCBOW 网络结构

这个网络结构的功能是——判断一句话是否是自然语言。怎么判断呢？使用的是概率，就是计算"一列词的组合"这句话的概率的连乘（联合概率）是多少，如果比较低，那么就可以认为不是一句自然语言，如果概率高，就是一句正常的话。

为描述方便，引入一些记号：

p^w：从根结点出发到达 w 对应结点的路径。

l^w：路径 p^w 中包含结点的个数。

$p_1^w, p_2^w, \cdots, p_{l^w}^w$：路径 p^w 中 l^w 个结点。

$d_2^w, d_3^w, \cdots, d_{l^w}^w \in \{0,1\}$：词 w 的霍夫曼编码，它由 $l^w - 1$ 位构成，d_i^w 表示路径 p^w 中第 i 个结点对应的编码（根结点不对应编码）。

$\boldsymbol{\theta}_1^w, \boldsymbol{\theta}_2^w, \cdots, \boldsymbol{\theta}_{l^w-1}^w \in \mathbf{R}^m$：路径 p^w 中非叶子结点对应的向量，$\boldsymbol{\theta}_i^w$ 表示路径 p^w 中第 i 个非叶子结点对应的向量。

图 10.5 显示了霍夫曼编码对应符号解释的例子。

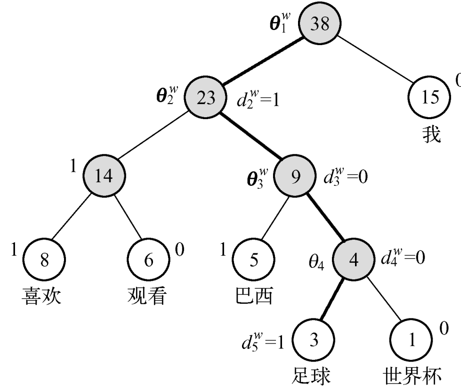


图 10.5　霍夫曼编码的例子

以图 10.5 为例解释引入的记号：w =“足球”，$l^w=5$，p_1^w、p_2^w、p_3^w、p_4^w、p_5^w 为路径 p^w 上的 5 个结点，d_2^w、d_3^w、d_4^w、d_5^w 分别为 1、0、0、1，即“足球”的霍夫曼编码 1001，$\boldsymbol{\theta}_1^w$、$\boldsymbol{\theta}_2^w$、$\boldsymbol{\theta}_3^w$、$\boldsymbol{\theta}_4^w$ 分别表示路径 p^w 上 4 个非叶子结点对应的向量。

那么，在图 10.5 所示网络结构下，如何计算 $p(w \mid Context(w))$？即如何利用向量 $\boldsymbol{x}_w$ 以及霍夫曼树计算 $p(w \mid Context(w))$。

以图 10.5 中词“足球”为例，从根结点到达“足球”这个叶子结点，中间经历了 4 次二分类，左为正类，右为负类。由于一个结点被分为负类的概率为

$$\sigma(\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta}) = \frac{1}{1+\mathrm{e}^{-\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta}}}$$

一个结点被分为正类的概率为

$$1-\sigma(\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta})$$

所以，第 1 次分类概率为

$$p(d_2^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_1^w) = 1-\sigma(\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta}_1^w)$$

第 2 次分类概率为

$$p(d_3^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_2^w) = \sigma(\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta}_2^w)$$

第 3 次分类概率为

$$p(d_4^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_3^w) = 1-\sigma(\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta}_3^w)$$

第 4 次分类概率为

$$p(d_5^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_4^w) = \sigma(\boldsymbol{x}_w^{\mathrm{T}}\boldsymbol{\theta}_4^w)$$

则有

$$p(足球 \mid Context(足球)) = \prod_{j=2}^{5} p(d_j^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_{j-1}^w)$$

应用 HCBOW，需要清楚以下问题：

（1）$p(w \mid Context(w))$ 计算

对字典 D 中的任意词 w，霍夫曼树必存在一条从根结点到词 w 的唯一路径 p^w，路径 p^w 上存在 $I^w - 1$ 的分支。将每个分支看作一次二分类，每次分类产生一个概率，将这些概率连乘起来就是所需的概率：

$$p(w \mid Context(w)) = \prod_{j=2}^{I^w} p(d_j^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_{j-1}^w)$$

其中

$$p(d_j^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_{j-1}^w) = \begin{cases} \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w), & d_j^w = 0 \\ 1 - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w), & d_j^w = 1 \end{cases}$$

或写成

$$p(d_j^w \mid \boldsymbol{x}_w, \boldsymbol{\theta}_{j-1}^w) = [\sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)]^{1-d_j^w} [1 - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)]^{d_j^w}$$

（2）参数修正

对数似然目标函数为

$$\begin{aligned} L(w,j) &= \sum_{w \in \mathbf{C}} \log p(w \mid Context(w)) \\ &= \sum_{w \in \mathbf{C}} \sum_{j=2}^{I^w} (1 - d_j^w) \log[\sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)] + d_j^w \log[1 - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)] \end{aligned}$$

所以，目标函数梯度为

$$\begin{aligned} \frac{\partial L(w,j)}{\partial \boldsymbol{\theta}_{j-1}^w} &= \frac{\partial}{\partial \boldsymbol{\theta}_{j-1}^w} \{(1 - d_j^w) \log[\sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)] + d_j^w \log[1 - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)\} \\ &= (1 - d_j^w)[1 - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)]\boldsymbol{x}_w - d_j^w \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)\boldsymbol{x}_w \\ &= [1 - d_j^w - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)]\boldsymbol{x}_w \end{aligned}$$

同理，利用 $L(w,j)$ 中 $\boldsymbol{x}_w$ 和 $\boldsymbol{\theta}_{j-1}^w$ 的对称性，有

$$\frac{\partial L(w,j)}{\partial \boldsymbol{x}_w} = [1 - d_j^w - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)]\boldsymbol{\theta}_{j-1}^w$$

利用随机梯度上升修正参数：$\boldsymbol{\theta}_j^w$ 的修正公式为

$$\boldsymbol{\theta}_{j-1}^w = \boldsymbol{\theta}_{j-1}^w + \eta[1 - d_j^w - \sigma(\boldsymbol{x}_w^{\mathrm{T}} \boldsymbol{\theta}_{j-1}^w)]\boldsymbol{x}_w$$

式中，$\boldsymbol{\eta}$ 为学习率。

$v(w)$ 的修正公式为

$$v(\widetilde{w}) = v(\widetilde{w}) + \eta \sum_{j=2}^{l^w} \frac{\partial L(w,j)}{\partial \boldsymbol{x}_w}, \quad \widetilde{w} \in Context(w)$$

由于 $\boldsymbol{x}_w$ 是 $v(w)$ 的和，所以要把 $\sum_{j=2}^{l^w} \frac{\partial L(w,j)}{\partial \boldsymbol{x}_w}$ 贡献到 $Context(w)$ 中每一个词向量上。

（3）伪代码

$e = 0$;

$x_w = \sum_{u \in Context(w)} v(u)$

FOR $j = 2$ TO l^w DO

{ $a = \sigma(x_w^{\mathrm{T}} \theta_j^w)$

$b = \eta(1 - d_j^w - a)$

$e := e + b\theta_{j-1}^w$

$\theta_{j-1}^w := \theta_{j-1}^w + bx_w$

}

FOR $u \in Context(w)$ DO

{ $v(u) := v(u) + e$ }

10.2.3 word2vec 流程

word2vec 算法流程如图 10.6 所示。

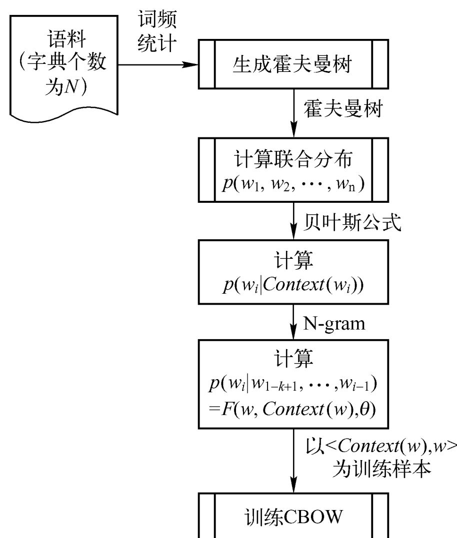


图 10.6　word2vec 算法流程

训练 CBOW 流程如图 10.7 所示。

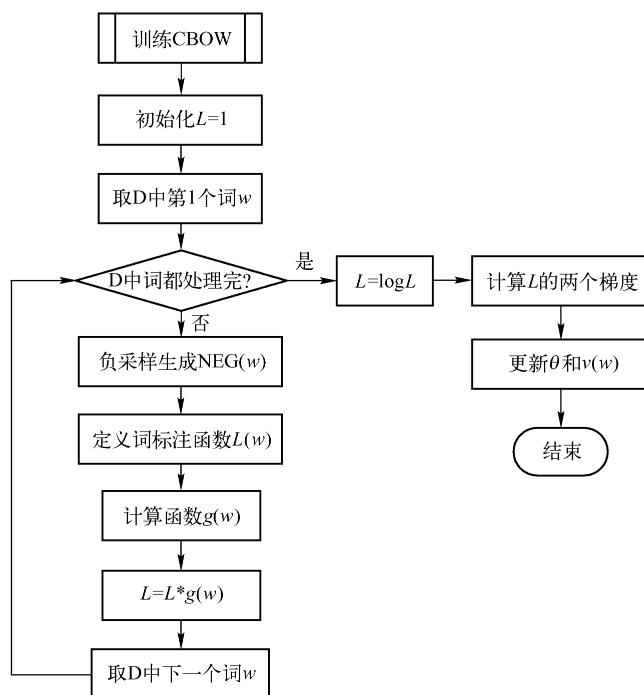


图 10.7　训练 CBOW 流程

10.3　word2vec 的 R 实现

10.3.1　tmcn. word2vec 包

tmcn. word2vec 包只有两个函数：一个是 word2vec；一个是计算单词之间 cos 距离的函数。这两个函数基本没什么附加的参数可以调节。

包的下载地址可见链接：https://r-forge.r-project.org/R/?group_id=1571。

```
> install.packages("tmcn.word2vec",repos="http://R-Forge.R-project.org")
```

包中有如下三个 demo 案例：

（1）第一个 demo

第一个是英文的 word2vec 方法的列举。

```
> require(tmcn.word2vec)
# English characters
```

```
> TrainingFile1 <- system.file("examples","rfaq.txt",package =
        "tmcn.word2vec")
> ModelFile1 <- file.path(tempdir(),"output","model1.bin")
> res1 <- word2vec(TrainingFile1,ModelFile1)
> dist1 <- distance(res1$model_file,"object")
> nrow(dist1)
```

使用 word2vec 之前，需要设置输入语料库文件（train_file）以及输出存放的文件（output_file）。其中 model1.bin 文件是二进制存储数据的方式。

distance 代表在 word2vec 所得到的词向量中，“object”词与其他单词之间的距离（一般采用 CosDist 距离）。该函数给出了与 object 最近的 20 个词语。

（2）第二个 demo

第二个 demo 是中文内容，以金庸的小说为输入预料。

```
> TrainingFile2 <- system.file("examples","xsfh_GBK.txt",package =
        "tmcn.word2vec")
> ModelFile2 <- file.path(tempdir(),"output","model2.bin")
> res2 <- word2vec(TrainingFile2,ModelFile2)
```

训练语料访问：

```
file./R-3.2.2/library/tmcn.word2vec/examples/xsfh_GBK.txt
Vocab size:1145
Words in train file:27824
Alpha:0.000000  Progress:790.77%  Words/thread/sec:22.69k  The model was generated
in C:/Users/long/AppData/Local/Temp/Rtmpm460Qd/ output !
> dist2 <- distance(res2 $ model_file,intToUtf8(c(33495,20154,20964)))
Word:?????   Position in vocabulary:316
> dist2
    Word    CosDist
1   可是   0.3918099
2     呼   0.2853436
3     踏   0.2702354
4     多   0.2539852
5   眼光   0.2498445
6     较   0.2495218
```

```
7      盒  0.2491589
8   群雄  0.2380990
9   后面  0.2367072
10     几  0.2354743
11  一只  0.2346973
12  一齐  0.2282183
13     夺  0.2276031
14     他  0.2262077
15 </s>  0.2241129
16  一招  0.2232552
17  双刀  0.2226523
18     救  0.2194796
19     学  0.2124971
20     即  0.2113202
```

代码解读：输入的语料库是分好词的，intToUtf8(c(33495,20154,20964))是一个词——“苗人凤”。经过运算得出了与其最近的词，看前20个词似乎没有什么特别有用的信息。

（3）第三个demo代码

```
# Chinese characters in UTF-8 encoding,for *nix
> TrainingFile3 <- system.file("examples","xsfh_UTF8.txt",package =
           "tmcn.word2vec")
> ModelFile3 <- file.path(tempdir(),"output","model3.bin")
> res3 <- word2vec(TrainingFile3,ModelFile3)
> dist3 <- distance(res3 $ model_file,intToUtf8(c(33495,20154,20964)))
```

10.3.2 word2vec自编译函数

自编译函数需要自己设置环境，只有两个函数：word2vec和distance。注意事项如下：

1）仔细阅读“.c.call extensions.pdf”文件，该文件详细写出了如何在计算机中搭建一个适用于R语言的二进制数据库。

2）Windows系统下，需要下载“Rtools.exe”文件，并改变环境变量的路径，同时重启计算机。

3）查看 train_word2vec. R，其中有在 R 中如何调用 word2vec 的 API。

具体的可以从 http://download. csdn. net/download/sinat_26917383/ 9513075 下载得到完整的自编译函数、说明以及上述提到的 PDF 文档。

- word2vec 函数

```
> word2vec <- function(train_file,output_file,
          binary = 1,             #输出格式,1—binary,0—txt
          cbow = 0,               #训练算法,0—n - gram 模型,1—词袋模型(CBOW)
          num_threads = 1,        #线程数
          num_features = 300,     #输出的词向量维数
          window = 10,            #窗口大小
          min_count = 40,         #最低词频,默认为 5
          sample = 1e - 3,        #下采样阈值(词频下限,建议 .00001 - .01)
          classes = 0)
```

除了上面所提到的参数，还有：

① alpha 表示学习速率。

② classes 表示词聚类簇的个数，从相关源码中可以得出，该聚类一般采用 k - means 算法，模型训练完成之后，得到了 . bin 这个词向量文件，文件的存储类型由 binary 参数决定，如果为 0，便可以直接用编辑器打开，进行查看。

- distance 函数

其中 word2vec 中提供了 distance 求词的 cosine 相似度，并排序。也可以在训练时，设置 - classes 参数来指定聚类的簇个数，使用 kmeans 进行聚类。

由于 word2vec 计算的是余弦值，距离范围为 0 ~ 1 之间，值越大代表这两个词关联度越高，所以越排在上面的词与输入的词越紧密。

```
distance(file_name,word,size)
```

输出是一个 list，然后可以得到 cos 距离。

10. 3. 3 使用 tmcn. word2vec 和 word2vec 注意的问题

1）word2vec 的自编译函数在使用时也需要加载 tmcn. word2vec 包，否则会出现以下 error：

```
Error in . C("CWrapper_word2vec",train_file = as. character(train_file),
```

```
:
C symbol name "CWrapper_word2vec" not in load table
```

2）tmcn. word2vec 与 word2vec 自编译互补。在 require（tmcn. word2vec）之后，可以直接调用 word2vec 函数，自编译函数也可以调节参数，而且有一个特殊的功能，可以自行聚类，并且可以通过 cbow =0 的参数选择使用 CBOW 模型还是 Skip – gram 模型，通过 binary =0 参数可以调整输出的是 txt 文件，而 tmcn. word2vec 包中输出只有 . bin 文件，难以读取。

3）tmcn. word2vec 与 word2vec 自编译中两个 word2vec 生成不一样的语料库，同时执行 distance 函数之后也计算不一样的词距离。语料库不同的原因，也许是因为 CBOW 模型与 Skip – gram 模型在抽取近邻词的时候采用随机抽样的方法，所以会产生不一样的结果。

4）最佳的使用 tmcn. word2vec 步骤：加载包（require（tmcn. word2vec））、执行自编译函数（word2vec/distance）、设定随机数（set. seed）这很关键，会影响输出结果）、用自编译函数来执行分析任务（选择模型、是否聚类、是否输出 txt 文件、词维度和词数量等）。

10. 4　学习指南

2013 年，Google 发布的 word2vec 工具是深度学习在自然语言领域的一项了不起的应用。基于 word2vec 还出现了 doc2vec。word2vec 只考虑了单词上下文的语义，但 doc2vec 不仅考虑了单词上下文的语义，还考虑了单词在段落中的顺序。

第 11 章　R 语言其他深度学习包

11.1　darch 包

darch 包是建立于 Hinton 和 Salakhutdinov 的 MATLAB 代码之上的，其实现方法包括了对比散度预训练算法和微调算法（如反向传播法或共轭梯度法）。

```
darch(x,                #输入数据矩阵或 data. frame(默认)
y                       #目标数据矩阵或 data. frame,依赖于 x 是矩阵还是 data. frame 而定
    layers = 10,        #每一层神经元数量
```

layers 是一个向量。默认是 $c(a,10,b)$，其中 a 是训练数据的列数；b 是目标的列数。如果这个长度为 1，那么它就作为隐藏层神经元的数量，而不是层的数量。

```
autosave = F,                                  #微调阶段是否自动保存激活的 darch 实例文件
autosave. epochs = round(darch. numEpochs/20),  #在多少次迭代后应该自动保存,
                                               #默认是 51,每次迭代完成 3 个微调,网络才会保存一次
autosave. dir = "./darch. autosave",           #自动保存实例文件的目录
bp. learnRate = 1,                             #反向传播的学习速率。当每一层使用不同的学习率,
                                               #长度要么为 1,要么等于权重矩阵
bp. learnRateScale = 1,                        #每次迭代后,学习速率乘以这个值
    bootstrap = F,                             #是否使用启动命令创建一个来自所给的训练数
                                               #据的训练和验证数据集
bootstrap. unique = T,
```

bootstrap. unique 表示是否采用特定的样本用于训练（默认为 TRUE1），如果取值 FALSE，则采用所有样本用于训练。采用所有样本将导致训练开销增大。注意：如果 bootstrap. num 大于 0，bootstrap. unique 选项将被忽略。

```
bootstrap.num = 0,                      #如果大于 0,将不重复抽取训练样本
cg.length = 2,                          #搜索的数量
cg.switchLayers = 1,                    #表明什么时候训练整个网络,而不是仅仅上面两层
darch = NULL,                           #是否要恢复 darch 实例
darch.batchSize = 1,                    #块的大小,即在权重更新之前呈递给网络的训练
                                        #样本的数量
darch.dither = F,                       #训练输入数据中是否包含数值列
darch.dropout = 0,                      #丢失率,如果一个元素丢失,输入的丢失就会被设
                                        #置成 0
darch.dropout.dropConnect = F,          #是否对隐藏层使用 DropConnect 而不是丢弃
darch.dropout.momentMatching = 0,       #丢弃期间进行迭代的次数,0 禁用时刻匹配
darch.dropout.oneMaskPerEpoch = F,      #每次迭代是否生成一个新的 mask
darch.elu.alpha = 1,                    #指数线性单元函数的 α 参数
darch.errorFunction = if(darch.isClass) crossEntropyError else mseError,
                                        #微调使用的误差函数
darch.finalMomentum = 0.9,              #微调阶段最后的动量
darch.fineTuneFunction = backpropagation,
```

darch.fineTuneFunction 表示微调函数。可能的取值包括 backpropag - ation（默认）、rpropagation、minimizeClassifier 和 minimizeAutoencoder（非监督学习）。

```
darch.initialMomentum = 0.5,            #微调阶段的初始动量
darch.isClass = T,                      #微调阶段输出是否应该被视作类标签
darch.maxout.poolSize = 2,              #当使用 maxout 激活函数时,池化的尺寸(maxout)
darch.maxout.unitFunction = linearUnit, #maxout 使用的初始值
darch.momentumRampLength = 1,
```

darch.momentumRampLength 参数与训练阶段的总体数量有关，即多少个阶段后，动量应该达到 darch.finalMomentum？值为 1 表明应该在最后时期达到 darch.finalMomentum，值为 0.5 表明应该在一半的训练完成后达到 darch.finalMomentum。注意，如果训练恢复到 darch.initialMomentum 和 darch.finalMomentum 相同的参数，这将导致动量坡道中的碰撞。在恢复训练时，可将 darch.momentumRampLength 设置为 0 以避免这一问题。

```
darch.nesterovMomentum = T,             #是否使用 Nesterov 加速
```

```
darch.numEpochs = 100,              #微调的次数
darch.returnBestModel = T,          #在训练的最后是否返回最好的模型,而不是在微调后
darch.returnBestModel.validationErrorFactor = 1 - exp( - 1),
```

当使用验证数据评估模型时，与训练错相比，验证错误应该被赋予多高的值？这个值介于0到1之间。默认情况下，这个值是$1-\frac{1}{e}$。训练误差因素和验证误差因素总会增加到1。因此，如果在这里超过了1，训练误差将会被忽略；如果在这里超过了0，验证误差将会被忽略。

```
darch.stopClassErr = - Inf,         #当分类错误低于该值,训练就停止
darch.stopErr = - Inf,              #当错误函数低于该值,训练就会停止
darch.stopValidClassErr = - Inf,    #当验证数据上的分类错误低于该值,训练就会停止
darch.stopValidErr = - Inf,         #当验证数据上的误差函数的值低于该值,训练就会停止
darch.trainLayers = T,              #是否训练所有层
darch.unitFunction = sigmoidUnit,
```

层1的长度数的层函数或者层向量函数。注意：第一个条目表明层1和层2之间的层函数，也就是层2的输出；层1没有层函数，因为输入值被直接使用。可能的函数包括linearUnit、sigmoidUnit、tanhUnit、rectifiedLinearUnit、softplusUnit、softmaxUnit和maxoutUnit。

```
darch.weightDecay = 0,              #权重衰减系数,默认是0
darch.weightUpdateFunction = weightDecayWeightUpdate,
```

权重更新函数或者权重向量更新函数darch.weightUpdateFunction与darch.unitFunction非常相似，权重更新函数包括weightDecayWeightUpdate和maxoutWeightUpdate。

注意：maxoutWeightUpdate必须在maxout激活函数之后使用。

```
dataSet = NULL,                 #来源于darch.DataSet(),可以手动指定
dataSetValid = NULL,            #DataSet包含验证数据的实例
generateWeightsFunction = generateWeightsGlorotUniform,
#层1长度数的权重产生函数或者层向量产生函数
gputools = F,                   #是否对矩阵乘法使用gputools
gputools.deviceId = 0,          #指定用于GPU混合乘法使用的设备
logLevel = NULL,
```

logLevel表示futile.logger日志级别。默认使用当前设定的日志级别，如果没有

被改变则是 futile. logger::flog. info。其他可能的等级（从最精简到最详细）包括 FATAL、ERROR、WARN、DEBUG 和 TRACE。

```
normalizeWeights = F,                        #是否归一化权重
normalizeWeightsBound = 15,                  #收益权重向量的 L2 norm 上界
preProc. factorToNumeric = F,                #是否把因子都要转换成数值型
preProc. factorToNumeric. targets = F,       #是否所有的因子都要转换成目标数据中的数值型
preProc. fullRank = T,                       #是否使用满秩编码
preProc. fullRank. targets = F,              #是否使用目标数据的满秩编码
preProc. orderedToFactor. targets = T,       #目标数据中的有序因子转换成无序因子
```

注意：有序因子通过 dummyVars 转换成数值型，而且不再用于分类任务。

```
preProc. params = F,                         #传递给 preProcess 函数的参数列表
preProc. targets = F,
```

preProc. targets 表示是否以目标数据为中心，是否扩展。与 preProc. params 不同，当预测新数据时，由于这一预处理必须恢复，所以这仅仅是一种用于目标数据使用和关闭的逻辑转换预处理。对于回归任务最有用。

注意：这将使原始网络倾斜。

```
rbm. allData = F,                            #是否将训练和验证数据用于预训练
rbm. batchSize = 1,                          #训练块的大小
rbm. consecutive = T,
```

rbm. consecutive 表示是否一次 rbm. numEpochs 阶段（TRUE，默认）训练一个 RBMs 或者一阶段交替训练每个 RBM（FALSE）。

```
rbm. errorFunction = mseError,
```

rbm. errorFunction 表示预训练期间的误差函数。这仅仅用于评估 RBM 误差，不影响训练本身。可能的误差函数包括 mseError 和 rmseError。

```
rbm. finalMomentum = 0. 9,                   #预训练期间的最后动量
rbm. initialMomentum = 0. 5,                 #预训练期间的初始动量
rbm. lastLayer = 0,
```

rbm. lastLayer 表明在哪一层停止预训练。可能的取值包括 0，意味着训练所有层；正整数，意味着 RBM 中 rbm. lastLayer 形成可见层之后停止训练；负整数，意味

着在来源于顶层 RBM 的 rbm. lastLayer RBM 停止训练。

```
rbm. learnRate = 1,              #预训练期间的学习速率
rbm. learnRateScale = 1,         #每个阶段后学习速率将会乘以这个值
rbm. momentumRampLength = 1,
```

rbm. momentumRampLength 参数表示多少个与 rbm. numEpochs 相关阶段后，动量应该达到 rbm. finalMomentum？值为 1 表明 rbm. finalMomentum 应该在最后阶段达到，值 0. 5 表明 rbm. finalMomentum 应该在训练完成一半时达到。

```
rbm. numCD = 1,                          #执行对比差异的完整步骤数,增加该值会大幅减缓
                                         #训练
rbm. numEpochs = 0,                      #预训练迭代次数
rbm. unitFunction = sigmoidUnitRbm,#预训练函数
rbm. updateFunction = rbmUpdate,         #预训练期间的更新函数
rbm. weightDecay = 2e - 04,              #预训练权重衰变
retainData = F,                          #训练之后是否存储 darch 实例中的训练数据
rprop. decFact = 0. 5,                   #减少训练因子,默认是 0. 6
rprop. incFact = 1. 2,                   #增加训练因子,默认是 1. 2
rprop. initDelta = 1/80,                 #更新的初始值,默认是 0. 0125
rprop. maxDelta = 50,                    #步长的上界,默认是 50
rprop. method = " iRprop + ",            #训练方法,默认是" iRprop + "
rprop. minDelta = 1e - 06,               #步长的下界,默认是 0. 000001
seed = NULL,                             #允许一个 通过 set. seed 设置的种子
shuffleTrainData = T,                    #在每个阶段之前是否对训练数据进行清洗
weights. max = 0. 1,                     #runif 函数的最大参数
weights. mean = 0,                       #rnorm 函数的平均参数
weights. min = - 0. 1,                   #runif 函数的最小参数
weights. sd = 0. 01,                     #rnorm 函数的 sd 参数
xValid = NULL,                           #用于验证的输入数据集
yValid = NULL                            #用于验证的目标数据,取决于 xValid 是数据矩阵或
                                         #数据框
)
```

【例 11. 1】 训练有效误差测试。

```
> library( darch)
```

```
> data( iris)
> model <- darch( Species ~ . ,iris,darch. errorFunction =
    " crossEntropyError" )
```

运行结果显示如下：

```
INFO [2017-01-02 09:15:32] Epoch:  96 of 100
INFO [2017-01-02 09:15:32] Classification error on Train set: 3.33% (5/150)
INFO [2017-01-02 09:15:33] Train set Cross Entropy error: 0.182
INFO [2017-01-02 09:15:33] Finished epoch  96 of 100 after 0.0891 secs (1683 patterns/sec)
INFO [2017-01-02 09:15:33] Epoch:  97 of 100
INFO [2017-01-02 09:15:33] Classification error on Train set: 5.33% (8/150)
INFO [2017-01-02 09:15:33] Train set Cross Entropy error: 0.211
INFO [2017-01-02 09:15:33] Finished epoch  97 of 100 after 0.0669 secs (2242 patterns/sec)
INFO [2017-01-02 09:15:33] Epoch:  98 of 100
INFO [2017-01-02 09:15:33] Classification error on Train set: 8% (12/150)
INFO [2017-01-02 09:15:33] Train set Cross Entropy error: 0.322
INFO [2017-01-02 09:15:33] Finished epoch  98 of 100 after 0.0739 secs (2030 patterns/sec)
INFO [2017-01-02 09:15:33] Epoch:  99 of 100
INFO [2017-01-02 09:15:33] Classification error on Train set: 5.33% (8/150)
INFO [2017-01-02 09:15:33] Train set Cross Entropy error: 0.191
INFO [2017-01-02 09:15:33] Finished epoch  99 of 100 after 0.0825 secs (1818 patterns/sec)
INFO [2017-01-02 09:15:33] Epoch: 100 of 100
INFO [2017-01-02 09:15:33] Classification error on Train set: 18% (27/150)
INFO [2017-01-02 09:15:33] Train set Cross Entropy error: 0.574
INFO [2017-01-02 09:15:33] Finished epoch 100 of 100 after 0.0819 secs (1877 patterns/sec)
INFO [2017-01-02 09:15:33] Classification error on Train set (best model): 1.33% (2/150)
INFO [2017-01-02 09:15:33] Train set (best model) Cross Entropy error: 0.170
INFO [2017-01-02 09:15:33] Best model was found after epoch 86
INFO [2017-01-02 09:15:33] Fine-tuning finished after 7.704 secs
```

11.2 Rdbn 包

11.2.1 Rdbn 原理

Rdbn 实现 R 环境的 RBM 和 DBN 的训练和学习。但目前还不能在 CRAN 上使用 Rdbn，只能在 github 上参考。

Rdbn 主要功能如下：

1）根据对比散度（Contrastive Divergence）预训练 DBN。

2）根据反向传播算法微调分类任务网络。

3）高级特征训练方法，例如动量加速学习和 L2 正则化。

4）基于 UNIX 系统 pthreads 线程实现并行处理。

11.2.2 Rdbn 安装

在 UNIX 操作系统下，可以简单输入命令：

```
R CMD installRdbn/
```

Windows 环境的 Rdbn 安装见例 8.1。

11.2.3 Rdbn 应用

【例 11.2】 基于 Vehicle 数据集训练分类器。

（1）加载依赖的包

```
>require(mlbench)
>data(Vehicle)
>require(Rdbn)
```

（2）最优神经网络协方差变换

```
>x<-t(Vehicle[,c(1:18)])
>y<-Vehicle[,19]
>for(i in c(1:(NCOL(Vehicle) -1))){
  x[i,]<-(Vehicle[,i] -min(Vehicle[,i]))/(max(Vehicle[,i]) -min(Vehicle[,i]))
}
```

（3）数据准备

```
set.seed(34)                    #参数不同结果不同
trainIndx<-sample(c(1:NCOL(x)),NCOL(x)*0.8,replace=FALSE)
testIndx<-c(1:NCOL(x))[!(c(1:NCOL(x))%in% trainIndx)]
```

（4）建模

```
>db<-dbn(x=x[,trainIndx],
      y=y[trainIndx],
      layer_sizes=c(18,100,150),
      batch_size=10,
      momentum_decay=0.9,
      learning_rate=0.1,
      weight_cost=1e-4,
      n_threads=8)
```

在训练网络时，重要的是每个 mini-batch 样本集要包含用于分类正例和反例，因为 Rdbn 不能改变样本的顺序。

如果每个 mini-batch 训练样本包含了所有类别的样本，就可以更换 Rdbn 的输

入顺序。下面脚本使用的函数 shuffle 返回的样本不排序，可以推广到多类情况。

shuffle 函数把样本后面的 50% 放到前面。

```
> shuffle <- function(n_elements) {
indx <- c(1:n_elements)
shuf <- c(which(indx %% 2 ==1),which(indx %% 2 ==0))
  return(order(shuf))
}
```

（5）模型部署

```
pred_dbn <- dbn.predict(db,data = x[ ,testIndx],n_threads = 8)
print(paste("% correct(dbn):",sum(pred_dbn == as.character(y[testIndx]))
/NROW(y[testIndx])))
```

另外，网络训练策略可以独立地应用，这提供了额外的控制训练参数，可以使模型有更好的性能。

```
> db <- dbn(layer_sizes = c(18,100,150),
     batch_size = 10,
     cd_n = 1,
     momentum_decay = 0.9,
     learning_rate = 0.1,
     weight_cost = 1e-4)
db <- dbn.pretrain(db,data = x[ ,trainIndx],n_epocs = 50,n_threads = 8)
```

微调使用带学习参数的反向传播算法。

```
db_refine <- dbn.refine(db,
     data = x[ ,trainIndx],
     labels = y[trainIndx],
     n_epocs = 100,
     rate_mult = 10,
     n_threads = 8)
pred_dbn <- dbn.predict(db_refine,data = x[ ,testIndx],n_threads = 8)
print(paste("% correct(dbn):",sum(pred_dbn == as.character(y[testIndx]))
     /NROW(y[testIndx])))
```

文件夹“Rdbn/test_functions”还有一个相关的例子。

11.3 H2O 包

11.3.1 H2O 原理

H2O 遵循多层前馈神经网络模型来进行预测性建模。本节详细地描述 H2O 的深度学习特征、参数配置，以及计算机实现。

（1）特征概要

H2O 的深度学习功能包括：

1）用于回归与分类任务的纯监督训练协议。

2）快速以及高效利用存储的 JAVA 实现，这种实现基于柱状压缩以及精细地图/微量递减。

3）在单结点或者一簇多重结点上进行多线程的分布式并行计算。

4）全自动的单神经元自适应学习速率，以求快速收敛。

5）可选的学习速率规格、退火以及动量选项。

6）正则化选项包括 L1、L2、dropout，Hogwild 以及平均化模型，这些选项用来防止过拟合。

……

（2）训练

1）初始化。多种深度学习架构结合了非监督预训练与监督训练，但是 H2O 使用单纯的监督训练协议。默认初始化方案是清一色的自适应选项，这些选项是基于网络规模进行了优化的。另一方面，可以选择服从均匀分布或者正态分布的随机初始化，对此，要说明换算因数。

2）激活函数与损失函数。可选的激活函数总结于表 11.1。表中，x_i 与 w_i 分别代表神经元的输入值和权重值；$\alpha = \sum_{i=1}^{n} w_i x_i + b$。

表 11.1　激活函数

激活函数	公式	范围
双曲正切	$f(\alpha) = \frac{e^{\alpha} - e^{-\alpha}}{e^{\alpha} + e^{-\alpha}}$	$f(\cdot) \in [-1,1]$
矫正线性	$f(\alpha) = \max(0,\alpha)$	$f(\cdot) \in \mathbf{R}_+$
极大输出	$f(\cdot) = \max(w_i x_i + b)$，如果 $f(\cdot) \geqslant 1$，则修正 $f(\cdot) = 1$	$f(\cdot) \in [-\infty,1]$

可选的损失函数列于表 11.2。

表 11.2　损失函数

损失函数	公　　式	典型应用
均方差	$L(\boldsymbol{W},\boldsymbol{B} \mid j) = \frac{1}{2} \Vert t^{(j)} - o^{(j)} \Vert_2^2$	回归
交叉熵	$L(\boldsymbol{W},\boldsymbol{B} \mid j) = \sum_{y=0} \ln(o_y^{(j)}) \cdot t_y^{(j)} + \ln(1 - o_j^{(j)}) \cdot (1 - j_y^j)$	分类

3）精调。使损失函数值最小化的过程类似于随机梯度下降（Stochastic Gradient Descent，SGD）。微调过程如下：

初始化：$\boldsymbol{W}$，$\boldsymbol{B}$。

迭代：

　　获取第 i 个训练样本；

　　修改 w_{jk} 和 b_{kj}

$$w_{jk} = w_{jk} - \alpha \frac{\partial L(\boldsymbol{W},\boldsymbol{B} \mid j)}{\partial w_{jk}}$$

$$b_{jk} = b_{jk} - \alpha \frac{\partial L(\boldsymbol{W},\boldsymbol{B} \mid j)}{\partial b_{jk}}$$

直到满足收敛条件。

其中，α 表示学习速率，它控制着梯度下降中的步长。

4）训练算法。在 H2O 中使用 SGD 进行分布式多线程训练，以下是一个算法概要：

步骤 1. 初始化全局模型参数 $\boldsymbol{W}$、$\boldsymbol{B}$。

步骤 2. 跨结点分布训练数据 T（结点可以不交，也可以重复）。

步骤 3. 迭代。

对训练子集 T_n 上的 n 个节点进行并行计算：

1）获取全局参数 W_n、B_n 的副本。

2）选择子集 $T_{na} \subset T_n$。

3）通过核 n_c 把 T_{na} 划分为 T_{nac}。

4）对节点 n 的核 n_c，并行计算：

　4.1）获取第 i 个训练样本，$i \in T_{nac}$。

　4.2）修改所有权重 $w_{jk} \in w_n$，偏移量 $b_{jk} \in B_n$。

$$w_{jk} = w_{jk} - \alpha \frac{\partial L(\boldsymbol{W},\boldsymbol{B} \mid j)}{\partial w_{jk}}$$

$$b_{jk} = b_{jk} - \alpha \frac{\partial L(\boldsymbol{W},\boldsymbol{B} \mid j)}{\partial b_{jk}}$$

$$\boldsymbol{W} = \underset{n}{\mathrm{Avg}} W_n$$

$$\boldsymbol{B} = \underset{\mathrm{n}}{\mathrm{Avg}} B_n$$

直到满足收敛条件。

步骤4. 根据验证集分数优化模型。

5）指定每步迭代的样本数。H2O 深度学习是可扩展的并且能够利用一大簇计算结点。一共有三种运行模式。默认行为是，让每个结点在整个数据集上训练，但是自动为每步迭代分配训练数据。

（3）正则化

H2O 的深度学习框架支持正则化技巧来防止过拟合。L1 与 L2 正则化方法都修改了损失函数为

$$L'(\boldsymbol{W},\boldsymbol{B} \mid j) = L(\boldsymbol{W},\boldsymbol{B} \mid j) + \lambda_1 R_1(\boldsymbol{W},\boldsymbol{B} \mid j) + \lambda_2 R_2(\boldsymbol{W},\boldsymbol{B} \mid j)$$

还有一种正则化方法称为 dropout。

（4）优化

H2O 在优化阶段，有手动模式与自动模式。手动模式特性包括动量训练和学习速率退火，而自动模式自适应调整学习速率。

动量训练：动量训练修改了反向传播，它允许前步迭代影响目前的更新。专门定义一个向量 $\boldsymbol{V}$ 来对更新进行修改如下：

$$v_{t+1} = \mu v_t - \alpha \nabla L(\theta_t)$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

其中，$\boldsymbol{\theta}$ 代表 $\boldsymbol{W}$、$\boldsymbol{B}$ 参数；μ 代表动量系数；α 代表学习速率。

（5）数据下载

当用 R 下载在使用 H2O 时需要的数据，这和通常的方法稍有些不同，即必须将数据集转化为 H2OParsedData 对象。例如，在 Http://bit. ly/1yywZzi. 下载天气数据，首先下载数据到 R 调试环境的当前工作目录，接着运行如下命令：

```
weather. hex = h20. uploadFile( h20_server. path = " weather. csv" ,header = true.
sep = ". " ,key = " weather. hex" )
```

快速概览数据，运行如下命令：

```
summary(weather.hex)
```

11.3.2 H2O 应用

【例 11.3】基于 H2O 的手写数字识别。

（1）加载 H2O 包

```
> install.packages("h2o",repos = (c("http://s3.amazonaws.com/h2o - release/
    h2o/rel - kahan/5/R",getOption("repos"))))
> library(h2o)
```

载入需要的 R：rjson、statmod 和 tools。

（2）启动 H2O

启动 H2O 获取连接对象' localH2O'：

```
> localH2O = h2o.init(ip = "localhost",port = 54321,startH2O = TRUE,
    Xmx = '1g')
```

为了停止 H2O，需执行：

```
< h2o.shutdown(localH2O)
```

H2O 启动后，就可以使用 http://localhost:54321。

（3）数据准备

下载训练集:http://www.pjreddie.com/media/files/mnist_train.csv;

下载测试集:http://www.pjreddie.com/media/files/mnist_test.csv。

```
> res <- data.frame(Training = NA,Test = NA,Duration = NA)
#加载数据到 H2O
> train_h2o <- h2o.importFile(localH2O,path = "C:/Users/jerry/Downloads/
    mnist_train.csv")
> test_h2o <- h2o.importFile(localH2O,path = "C:/Users/jerry/Downloads/
    mnist_test.csv")
> y_train <- as.factor(as.matrix(train_h2o[,1]))
> y_test <- as.factor(as.matrix(test_h2o[,1]))
```

（4）建模

训练模型要很长一段时间，最后一行有相应的进度条可查看。

```
> model <- h2o. deeplearning( x = 2 :785,              #输入变量个数
                             y = 1,                    #响应变量个数
                             data = train_h2o,
                             activation = "Tanh",
                             balance_classes = TRUE,
                             hidden = c(100,100,100),  ##3 层隐藏层
                             epochs = 100)
```

输出模型结果：

```
< model
```

IP Address:localhost

Port:54321

Parsed Data Key:mnist_train. hex

Deep Learning Model Key:DeepLearning_9c7831f93efb58b38c3fa08cb17d4e4e

Training classification error:0

Training mean square error:Inf

Validation classification error:0

Validation square error:Inf

Confusion matrix:

Reported onmnist_train. hex

Predicted

Actual	0	1	2	3	4	5	6	7	8	9	Error
0	5923	0	0	0	0	0	0	0	0	0	0
1	0	6742	0	0	0	0	0	0	0	0	0
2	0	0	5958	0	0	0	0	0	0	0	0
3	0	0	0	6131	0	0	0	0	0	0	0
4	0	0	0	0	5842	0	0	0	0	0	0
5	0	0	0	0	0	5421	0	0	0	0	0
6	0	0	0	0	0	0	5918	0	0	0	0
7	0	0	0	0	0	0	0	6265	0	0	0
8	0	0	0	0	0	0	0	0	5851	0	0
9	0	0	0	0	0	0	0	0	0	5949	0
tals	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949	0

(5) 模型评估

```
> yhat_train <- h2o.predict(model,train_h2o) $ predict
> yhat_train <- as.factor(as.matrix(yhat_train))
> yhat_test <- h2o.predict(model,test_h2o) $ predict
> yhat_test <- as.factor(as.matrix(yhat_test))
```

查看前 100 条预测与实际的数据相比较

```
< y_test[1:100]
```

```
[1]7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3
5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4
[67]6 4 3 0 7 0 2 9 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9
Levels:0 1 2 3 4 5 6 7 8 9
```

```
> yhat_test[1:100]
```

```
[1]7 2 1 0 4 1 8 9 4 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7 1 2 1 1 7 4 2 3
5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4
[67]6 4 3 0 7 0 2 9 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9
Levels:0 1 2 3 4 5 6 7 8 9
```

查看并保存结果:

```
> library(caret)
> res[1,1] <- round(h2o.confusionMatrix(yhat_train,y_train) $ overall[1],4)
> res[1,2] <- round(h2o.confusionMatrix(yhat_test,y_test) $ overall[1],4)
> print(res)
```

11.4 deepnet 包

R 的伟大之处在于，它有多个估算神经网络模型的包。建议读者使用多种多样的包和学习算法来构建模型。

deepnet 包实现了一些 DNN 结构和神经网络相关算法，包括 BP、RBM 训练、DBN、AE、CNN 和 RNN 算法等。

【例 11.4】 使用 deepnet 包建立一个传统的反向传播 DNN。

（1）加载所需的包

```
>library(deepnet)
```

（2）设置变量

使用 set. seed 方法保证程序的可重复性，并将属性变量存储在 R 对象 X 中，响应变量存储在 R 对象 Y 中。

```
>set. seed(2016)
>X = data[train,1:9]
>Y = data[train,10]
```

（3）建立 DNN

```
>fitB <- nn. train(x = X,y = Y,
    initW = NULL,
    initB = NULL,
    hidden = c(10,12,20),
    learningrate = 0. 58,
    momentum = 0. 74,
    learningrate_scale = 1,
    activationfun = "sigm",        #激活函数
    output = "linear",             #输出神经元使用线性激活函数
    numepochs = 970,
    batchsize = 60,
    hidden_dropout = 0,
    visible_dropout = 0)
```

这些语句与以前看到的非常相似，然而，仍旧要逐行地看一下。神经网络存储在 R 对象 fitB 中。注意，这里使用语句 x = X，y = Y 传递属性变量和响应变量。deepnet 包允许指定神经元权重（initW）和偏差（initB）的起始值；将这两个值都设置为 NULL，以便算法随机选择它们的值。此处的 DNN 有三个隐藏层，在第一、第二和第三隐藏层中分别具有 10、12 和 20 个神经元。

若要使用反向传播算法，必须指定学习率和动量。学习率控制神经网络收敛的速度。简而言之，动量在梯度下降更新中增加了上一梯度的加权平均。它倾向于抑制噪声，特别是在误差函数的高曲率区域中。因此，动量可以帮助网络避免陷入局部最小值。这三个参数一般通过反复试验来设置，学习率、动量和学习率标度分别

选择 0.58、0.74 和 1。

接下来的两行指定了隐藏神经元和输出神经元的激活函数。对于隐藏神经元使用逻辑函数 sigm；其他选项则包括 linear 或 tanh。对于输出神经元使用线性激活函数，其他选项则包括 sigm 和 softmax。该模型迭代超过 970 次，每次块的大小为 60。在输入层或隐藏层没有 Dropout 神经元。

（4）模型部署

```
> Xtest <- data[ -train,1:9]
> predB <- nn. predict(fitB,Xtest)
```

性能指标计算如下：

```
< round(cor(predB,data[ -train,10])^2,6)
```

```
[,1]
[1,]0.930665
```

```
> mse(data[ -train,10],predB)
```

```
[1]0.08525447959
> rmse(data[ -train,10],predB)
[1]0.2919836975
```

总的来说，使用 deepnet 包构建模型的过程与使用神经网络包的过程非常相似。这是使用 R 的最大的优点，包通常以类似的方式工作（当然要指定的参数可能有所不同）。这种灵活性可以使我们能够使用各种不同的学习算法和调整参数来快速构建类似的 DNN 模型。

DNN 常常具有相同的拓扑，但是不同的学习算法或调整参数将在相同的底层数据上以不同的方式执行。还有重要一点不能忘记，选择最佳 DNN 模型的过程需要选择拓扑、神经元、层数、学习算法和调整参数，这使得模型的组合相当复杂。尽管如此，已经看到，强大的 DNN 模型可以在 R 中被快速构建、训练和测试。这能够使得它在图像、声音和信号处理等传统领域之外的各学科的使用异常繁荣。

11.5 mbench 包

目前为止，绝大多数已发布的 DNN 的应用都涉及数据分类问题。这些应用绝大

多数涉及图像和信号处理。本节讨论 DNN 模型在健康相关的领域上的应用。通过这个案例研究，可以放开思路，结合相关专业知识，将这个应用转化到科研领域中。

【例 11.5】使用 mbench 包中的 PimaIndiansDiabetes2 数据构建 DNN 模型。

（1）数据集

数据集与糖尿病、消化和肾脏疾病研究有关，它包含 768 个观测值，每个观察值有 9 个变量，这些变量来源于 21 岁以上的印度传统 Pima 女性。表 11.3 是对数据集变量的描述。

```
<data("PimaIndiansDiabetes2",package = "mlbench")   #数据加载
>ncol(PimaIndiansDiabetes2)                          #列数
```

```
[1]9
```

```
>nrow(PimaIndiansDiabetes2)                            #行数
```

```
[1]768
```

表 11.3　PimaIndiansDiabetes2 数据框

变　量　名	描　　述
pregnant	怀孕次数
glucose	血浆葡萄糖浓度
pressure	舒张血压（mmHg）
triceps	肱三头肌皮肤褶皱厚度（mm）
insulin	2 小时内血清含胰岛素数（MU/ml）
mass	身体质量指数
pedigree	糖尿病血流函数
age	年龄
diabetes	对糖尿病的测试（分类变量，阴性/阳性）

diabetes 为响应变量。注意到，压力、肱三头肌、胰岛素和质量包含 NA，这些是缺失值。收集数据时可能面临着丢失数据的问题，如人们不愿或忘记回答问题，因此，数据常常丢失或者无法被正确记录。这里数据似乎缺失很多，最好检查一下实际缺失的数据。

```
<apply(PimaIndiansDiabetes2,2,function(x)sum(is.na(x)))
```

```
pregnant  glucose  pressure  triceps  insulin  mass  pedigree  age  diabetes
    0        5        35       227      374     11      0       0      0
```

数据存在大量缺失值，特别是胰岛素和三头肌的属性。应该如何处理这个问题？

最常用的也是最容易的方法是仅使用那些有完整个人信息的数据。用一个合理的值估算缺失的观测值也是一个替代方法。例如，可以用属性均值或中数替换 NA。更复杂的方法是使用分布模型（例如最大似然和多重插补）。

由于胰岛素和肱三头肌缺失太多，从样本中删除这两个属性，并使用 na. omit 方法删除剩下的缺失值。被删除的数据存储在 temp 中。

```
<temp <-(PimaIndiansDiabetes2)
<temp $ insulin <-NULL
<temp $ triceps <-NULL
<temp <-na. omit(temp)
```

要做出有意义的分析，必须拥有足够的观测值。因此，考查一下数据：

```
>nrow(temp)
```

```
[1]724
```

```
>ncol(temp)
```

```
[1]7
```

数据包含了 724 个数据，响应变量存储在 R 对象 y 中，并从 temp 中删除它。注意，现在矩阵 temp 中只包含变量属性。使用缩放的方法将属性数据标准化；然后将 y（作为因子）合并到 temp 中。

```
<y <-(temp $ diabetes)
<temp $ diabetes <-NULL
<temp <-scale(temp)
<temp <-cbind(as. factor(y),temp)
```

继续检查一下，以确保这个类是一个矩阵类型：

```
<class(temp)[1]"matrix"
```

也可以使用 summary，查看数据摘要：

```
>summary(temp)
```

```
      vi              pregnant                glucose                pressure
 Min.   :1.000000   Min.   :-1.1496428   Min.   :-2.5327649   Min.   :-3.90961708
 1st Qu.:1.000000   1st Qu.:-0.8522718   1st Qu.:-0.7197585   1st Qu.:-0.67856547
 Median :1.000000   Median :-0.2575298   Median :-0.1587835   Median :-0.03235514
```

```
Mean     :1.343923   Mean     :  0.0000000   Mean     :  0.0000000   Mean     :  0.00000000
3rd Qu.:2.000000   3rd Qu.:  0.6345831   3rd Qu.:  0.6542239   3rd Qu.:  0.61385518
Max.     :2.000000   Max.     :  3.9056640   Max.     :  2.5078806   Max.     :  4.00645938
      mass                  pedigree                  age
Min.     :-2.071018931   Min.     :-1.1939431   Min.     :-1.0497592
1st Qu.  :-0.721029128   1st Qu.  :-0.6914078   1st Qu.  :-0.7947740
Median   :-0.009744179   Median   :-0.2881760   Median   :-0.3697989
Mean     :  0.000000000   Mean     :  0.0000000   Mean     :  0.0000000
3rd Qu.:  0.599928635   3rd Qu.:  0.4596085   3rd Qu.:  0.6501416
Max.     :  5.027314546   Max.     :  5.8535871   Max.     :  4.0499431
```

最后，选择训练样本（724 个观测值中的 600 个）。

```
< set.seed(2016)
< n = nrow(temp)
< n_train <- 600
< n_test <- n - n_train
< train <- sample(1:n,n_train,FALSE)
```

（2）使用 RSNNS 包建立 DNN

```
< library(RSNNS)
```

为让问题尽可能简单，将响应变量赋值给 Y，属性变量赋值给 X。

```
< set.seed(2016)
< X <- temp[train,1:6]
< Y <- temp[train,7]
```

现在的问题是，如何在 RSNNS 包中指定分类 DNN。

```
> fitMLP <- mlp(x = X,y = Y,
    size = c(12,8),
    maxit = 1000,
    initFunc = "Randomize_Weights",
    initFuncParams = c( -0.3,0.3),
    learnFunc = "Std_Backpropagation",
    learnFuncParams = c(0.2,0),
    updateFunc = "Topological_Order",
    updateFuncParams = c(0),
    hiddenActFunc = "Act_Logistic",
```

```
shufflePatterns = TRUE,
linOut = TRUE)
```

这个网络有两个隐藏层；第一个隐藏层包含 12 个神经元；第二隐含层包含 8 个神经元。使用隐藏层中的逻辑激活函数和输出层神经元的线性激活函数来随机地初始化权重和偏差。

注意到，对于这个数据集，R 代码执行相对较快。

（3）使用 predict 函数进行预测

```
>predMLP <- sign(predict(fitMLP,temp[-train,1:6]))
```

既然有了分类数据，就应该查看其混淆矩阵。

```
<table(predMLP,sign(temp[-train,7]),dnn = c("Predicted","Observed"))
```

```
          Observed
Predicted  -1   1
       -1  67   9
        1  21  27
```

注意：模型不能完美地拟合数据。混淆矩阵中的对角线元素表示分类错误的数据数量。将错误分类的数据计算为错误率通常是有帮助的。做法如下：

```
<error_rate = (1 - sum(predMLP == sign(temp[-train,7]))/124)
<round(error_rate,3)
```

```
[1]0.242
```

分类 DNN 具有大约 24% 的总体错误率（或大约 76% 的准确率）。

要强调的是，使用 R 可以轻松地、快速地构建分类 DNN。如果使用 C++ 和 C 编写代码要许多行，才可以完成现在 R 中几行代码就可以实现事情。如果读者对数据科学感兴趣，并且希望使用相关学科的工具来提取有意义的信息，那么使用 R 肯定会事半功倍。

11.6 AMORE 包

nnet 包提供了最常见的前馈反向传播神经网络算法，AMORE 包则更进一步提供了更为丰富的控制参数，并可以增加多个隐藏层。

【例 11.6】使用 AMORE 包构建一个分类 DNN。

（1）分离 RSNNS 包并加载 AMORE

```
< detach("package:RSNNS",unload = TRUE)
< library(AMORE)
```

（2）建模

```
> net <- newff(n.neurons = c(6,12,8,1),
               learning.rate.global = 0.01,
               momentum.global = 0.5,
               error.criterium = "LMLS",
               Stao = NA,
               hidden.layer = "sigmoid",
               output.layer = "purelin",
               method = "ADAPTgdwm")
```

绝大部分包的用法都类似于之前看到的。但需注意，AMORE 包需要指定输入和输出结点的数量以及隐藏结点。这里使用语句 n. neurons = c(6,12,8,1)来实现，第一个数字反映了六个输入属性，第二个和第三个值表示第一和第二隐藏层中神经元的数量，并且在这种情况下最终输出神经元的数量为 1。

多数情况下，数据科学家通过最小化训练集的均方差来训练 DNN。然而，当出现异常值时，训练得到的模型可能难以捕获生成数据的机制。为此，在构建 DNN 模型时，希望最小化鲁棒的误差度量。建议使用平均对数平方误差（LMLS）。AMORE 软件包中的其他鲁棒选择包括 Tao error60 的“TAO”以及标准均方误差（LMS），参数方法指定使用的学习方法，而本例选择使用自适应动量梯度下降的方法——ADAPTgdwm。

（3）数据集

将属性变量赋值给 R 对象 X，将响应变量赋值给 R 对象 Y。

```
< X <- temp[train,]
< Y <- temp[train,7]
```

（4）数据拟合

- 训练集拟合

```
> fit <- train(net,P = X,T = Y,error. = "LMLS",report = TRUE,
          show.step = 100,n.shows = 5)
```

模型运行时，将看到如下的输出：

```
index. show :1 LMLS 0.239138435481238
index. show :2 LMLS 0.236182280077741
index. show :3 LMLS 0.230675203275236
index. show :4 LMLS 0.222697557309232
index. show :5 LMLS 0.214651839732672
```

- 测试集拟合

一旦模型开始收敛，立即使用 sim 算法基于测试样本拟合模型。

```
< pred <- sign(sim(fit $ net,temp[ -train,]))
< table(pred,sign(temp[ -train,7]),dnn = c("Predicted","Observed"))
```

```
              Observed
Prdeicted    -1     1
       -1    71    10
        1    17    26
```

（5）模型部署

通过误差率评估模型：

```
< error_rate = (1 - sum(pred == sign(temp[ -train,7]))/124)
< round(error_rate,3)
```

```
[1]0.218
```

总体上，具有动量学习的自适应梯度下降结合鲁棒误差的方法，相对于使用 RSNNS 包建立的分类 DNN 有较低的误分类率，其误分类率约为 22%。这里的关键是，当建立 DNN 模型时，要使用不同的包、不同的学习算法、不同的网络架构等。换言之，构建卓越的 DNN 就是通过实验。

11.7 学习指南

本章介绍了用于深度学习的其他 6 个 R 包。实际应用要根据目标、数据来选择相应的 R 包。

附　　录

附录 A　深度学习发展史

深度学习起点是神经认知机模型，此时已经出现了卷积结构，经典的 LeNet 诞生于 1998 年。然而之后卷积神经网络（CNN）的锋芒开始被支持向量机（SVM）等手工设计的特征盖过。随着 ReLU 和 Dropout 的提出，以及 GPU 和大数据带来的历史机遇，CNN 在 2012 年迎来了历史突破——AlexNet（见图 A. 1）。

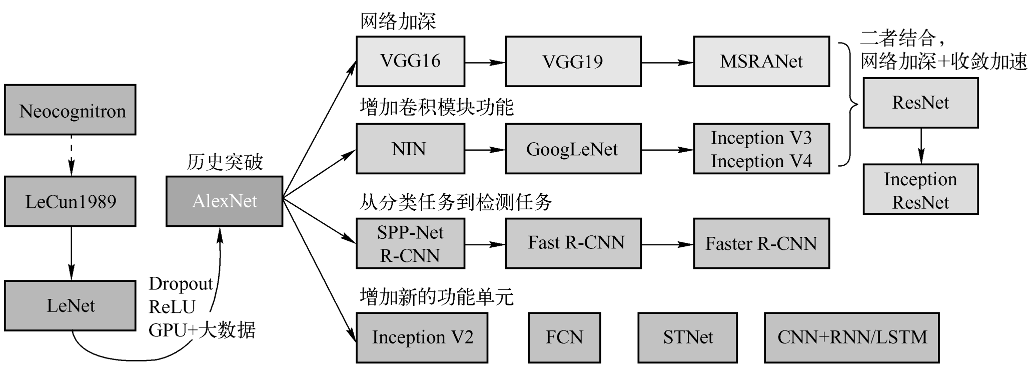


图 A. 1　神经网络演化过程

A. 1　LeNet5

LeNet5 是最早的卷积神经网络之一，并且推动了深度学习领域的发展。自从 1998 年开始，在许多次成功的迭代后，这项由 Yann LeCun 完成的开拓性成果被命名为 LeNet5，其架构见图 A. 2，麻雀虽小，但五脏俱全，卷积层、pooling 层、全连接层，这些都是现代 CNN 网络的基本组件（见第 3 章）。

输入尺寸：32 ×32；

卷积层：3 个；

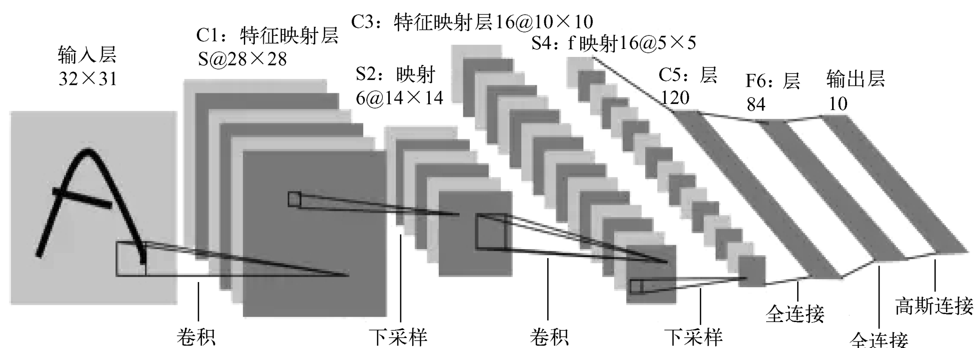


图 A. 2　LeNet5 架构

下采样层：2 个；

全连接层：1 个；

输出层：10 个数字（0 ~ 9）的概率。

可以说 LeNet 是 CNN 的开端，所以这里简单介绍 LeNet 各个组件的用途与意义。

（1） Input(32 * 32)

输入图像大小为 32 × 32。这要比 mnist 数据库中最大的字母（28 × 28）还大。这样做的目的是希望潜在的明显特征，如笔画断续、角点能够出现在最高层特征监测子感知的中心。

（2） C1,C3,C5(卷积层)

卷积核在二维平面上平移，并且卷积核的每个元素与被卷积图像对应位置相乘，再求和。通过卷积核的不断移动，就有了一个新的图像，这个图像完全由卷积核在各个位置时的乘积求和的结果组成。

二维卷积在图像中的效果就是：对图像的每个像素的邻域（邻域大小就是核的大小）加权求和得到该像素点的输出值。具体做法见图 3. 2。

卷积运算一个重要的特点就是：通过卷积运算可以使原信号特征增强，并且降低噪声。

不同的卷积核能够提取到图像中的不同特征，下面是不同卷积核得到的不同的卷积特征图（见图 A. 3、A. 4）。

以 C1 层进行说明：C1 层是一个卷积层，有 6 个卷积核（提取 6 种局部特征），核大小为 5 × 5，能够输出 6 个特征图（Feature Map），大小为 28 × 28。C1 有 156 个可训练参数（每个滤波器 5 × 5 = 25 个 unit 参数和 1 个 bias 参数，一共 6 个滤波器，

共(5×5+1)×6=156 个参数)，共 156×(28×28)=122 304 个连接。

rect
big rect
gaussian
sharpen
edges
shift
hand shake
custom

0.01	0.02	0.03	0.02	0.01
0.02	0.06	0.08	0.06	0.02
0.03	0.08	0.11	0.08	0.03
0.02	0.06	0.08	0.06	0.02
0.01	0.02	0.03	0.02	0.01

normalize

rect
big rect
gaussian
sharpen
edges
shift
hand shake
custom

0.00	0.00	0.00	0.00	0.00
0.00	0.00	−2.00	0.00	0.00
0.00	−2.00	9.00	−2.00	0.00
0.00	0.00	−2.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00

normalize

图 A. 3　卷积特征图 1

rect
big rect
gaussian
sharpen
edges
shift
hand shake
custom

0.00	0.00	0.00	0.00	0.00
0.00	0.00	−2.00	0.00	0.00
0.00	+2.00	8.00	−2.00	0.00
0.00	0.00	−2.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00

normalize

rect
big rect
gaussian
sharpen
edges
shift
hand shake
custom

0.03	0.04	0.04	0.03	0.05
0.01	0.03	0.00	0.00	0.00
0.04	0.07	0.03	0.03	0.12
0.05	0.08	0.00	0.03	0.03
0.06	0.07	0.15	0.00	0.03

normalize

图 A. 4　卷积特征图 2

(3) S2,S4(pooling 层)

S2、S4 是下采样层，是为了降低网络训练参数及模型的过拟合程度。池化/采样的方式通常有以下两种：

1） Max－Pooling：选择 Pooling 窗口中的最大值作为采样值。

2） Mean－Pooling：将 Pooling 窗口中的所有值相加取平均，以平均值作为采样值。

S2 层是 6 个 14×14 的特征图，图中的每一个单元与上一层的 2×2 领域相连接，所以，S2 层是 C1 层的 1/4。

（4） F6（全连接层）

F6 是全连接层，类似 MLP 中的一个 Layer，共有 84 个神经元（为什么选这个数字？与输出层有关），这 84 个神经元与 C5 层进行全连接，所以需要训练的参数是：（120＋1）×84＝10 164。

如同经典神经网络，F6 层计算输入向量和权重向量之间的点积，再加上一个偏置。然后将其传递给 Sigmoid 函数产生单元 i 的一个状态。

（5） Output（输出层）

输出层由欧氏径向基函数（Euclidean Radial Basis Function）单元组成，每类一个单元，每个有 84 个输入。

换句话说，每个输出 RBF 单元计算输入向量和参数向量之间的欧氏距离。输入离参数向量越远，RBF 输出的值越大。用概率术语来说，RBF 输出可以被理解为 F6 层配置空间的高斯分布的负 log－likelihood。给定一个输式，损失函数应能使得 F6 的配置与 RBF 参数向量（即模式的期望分类）足够接近。图 A. 5 显示在两个数据上的测试结果。



图 A. 5　测试结果

LeNet5 的架构基于这样的观点：（尤其是）图像的特征分布在整张图像上，以及带有可学习参数的卷积是一种用少量参数在多个位置上提取相似特征的有效方式。在那时候，没有 GPU 帮助训练，甚至 CPU 的速度也很慢。因此，能够保存参数以及计算过程是一个关键进展。这和将每个像素用作一个大型多层神经网络的单独输入相反。LeNet5 阐述了那些像素不应该被使用在第一层，因为图像具有很强的空间相关性，而使用图像中独立的像素作为不同的输入特征则利用不到这些相关性。

（6）LeNet5 特征

1）卷积神经网络使用 3 个层作为一个序列：卷积、池化和非线性。这可能是图像深度学习的关键特征！

2）使用卷积提取空间特征。

3）使用映射到空间均值下采样（Subsample）。

4）双曲正切（tanh）或 S 型（Sigmoid）形式的非线性。

5）多层神经网络（MLP）作为最后的分类器。

6）层与层之间的稀疏连接矩阵以避免大的计算成本。

总体来看，LeNet5 网络是最近大量架构的起点，并且也给这个领域的许多应用带来了灵感。

1998～2010 年神经网络处于孵化阶段。大多数人没有意识到它们不断增长的力量，与此同时其他研究者则进展缓慢。由于手机相机以及便宜的数字相机的出现，越来越多的数据可被利用。并且计算能力也在成长，CPU 变得更快，GPU 变成了多种用途的计算工具。这些趋势使得神经网络有所进展，虽然速度很慢。数据和计算能力使得神经网络能完成的任务越来越有趣。之后一切变得清晰起来……

A.2 Dan Ciresan Net

2010 年，DanClaudiu Ciresan 和 Jurgen Schmidhuber 发布了最早的 GPU 神经网络的一个实现。这个实现是在一块 NVIDIA GTX 280 图形处理器上运行 9 层的神经网络，包含前向与反向传播。

A.3 AlexNet

2012 年，AlexKrizhevsky 发表了 Alexet（参见：ImageNet Classification with Deep Convolutional Neural Networks），它是 LeNet 的一种更深更宽的版本，并以显著优势赢得了困难的 ImageNet 竞赛，其架构如图 3.6 所示。

AlexNet 将 LeNet 的思想扩展到了更大的能学习更复杂的对象与对象层次的神经

网络上。

（1） AlexNet 特征

1） 使用修正的线性单元（ReLU） 作为非线性。

2） 在训练的时候使用 Dropout 技术有选择地忽视单个神经元，以避免模型过拟合。

3） 覆盖进行最大池化，避免平均池化的平均化效果。

4） 使用 GPU NVIDIA GTX 580 减少训练时间。

在那时，GPU 相比 CPU 可以提供更多数量的核，训练效率可以提升 10 倍，这又反过来允许使用更大的数据集和更大的图像。

AlexNet 可以说是具有历史意义的一个网络结构，可以说在 AlexNet 之前，深度学习已经沉寂了很久。历史的转折在 2012 年到来，AlexNet 在当年的 ImageNet 图像分类竞赛中，top－5 错误率比上一年的冠军下降了 10 个百分点，而且远远超过当年的第二名。

（2） AlexNet 原理

1） 样本倍增。

有一种观点认为神经网络是靠数据喂出来的，若增加训练数据，则能够提升算法的准确率，因为这样可以避免过拟合，而避免了过拟合就可以增大网络结构。当训练数据有限的时候，可以通过一些变换来从已有的训练数据集中生成一些新的数据，进而扩大训练数据的规模。

其中，最简单、通用的图像数据变形的方式如下（见图 A. 6）：

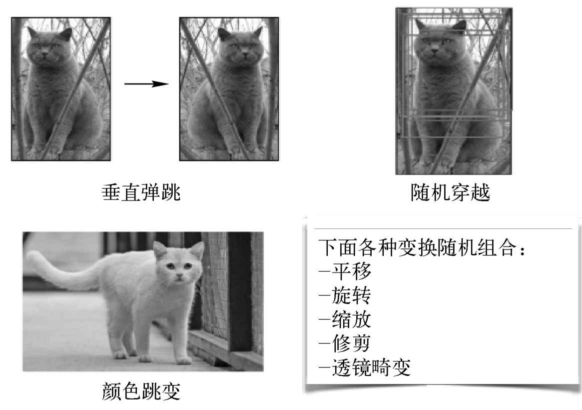


图 A. 6　AlexNet 应用结果

① 从原始图像（256，256）中，随机地 crop 出一些图像（224，224）。【平移变换，crop】

② 水平翻转图像。【反射变换，flip】

③ 给图像增加一些随机的光照。【光照、彩色变换，color jittering】

AlexNet 训练的时候，在样本倍增上处理得很好：

① 随机 crop。训练时候，对于 256 × 256 的图片进行随机 crop 到 224 × 224，然后允许水平翻转，那么相当于将样本倍增到 $(256-224)^2 \times 2 = 2048$。

② 测试的时候，对左上、右上、左下、右下、中间做了 5 次 crop，然后翻转，共 10 次 crop，之后对结果求平均。若不做随机 Crop，则大网络基本都过拟合（Under Substantial Overfitting）。

③ 对 RGB 空间做主成分分析（PCA），然后对主成分做一个（0，0.1）的高斯扰动。结果使错误率下降了 1%。

2）ReLU 激活函数。

Sigmoid 是常用的非线性的激活函数，它能够把输入的连续实值“压缩”到 0 ~ 1 之间。特别地，如果是非常大的负数，那么输出就是 0；如果是非常大的正数，输出就是 1。

但是 Sigmoid 有一些致命的缺点：

当输入非常大或者非常小的时候，会有饱和现象，这些神经元的梯度是接近于 0 的。如果初始值很大，则梯度在反向传播的时候因为需要乘上一个 Sigmoid 的导数，所以会使得梯度越来越小，这会导致网络变得很难学习。

Sigmoid 的输出不是 0 均值，这是不可取的，因为这会导致后一层的神经元将得到上一层输出的非 0 均值的信号作为输入。

产生的一个结果就是：如果数据进入神经元的时候是正的（若 $\boldsymbol{x} > 0, f = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$），那么 $\boldsymbol{w}$ 计算出的梯度也会始终都是正的。

当然，如果按 Batch 去训练，那么这个 Batch 可能得到不同的信号。因此，非 0 均值这个问题虽然会产生一些不好的影响，但与上面提到的梯度变小问题相比要好很多。

ReLU 的数学表达式如下：

$$f(x) = \max(0, x)$$

很显然，从图 A.7a 可以看出，输入信号小于 0 时，输出都是 0；大于 0 的情况下，输出等于输入。在 $\boldsymbol{w}$ 是二维的情况下，使用 ReLU 之后的效果如图 A.7 所示。

Alex 用 ReLU 代替了 Sigmoid，发现使用 ReLU 得到的 SGD 的收敛速度比 Sigmoid/tanh 快很多。

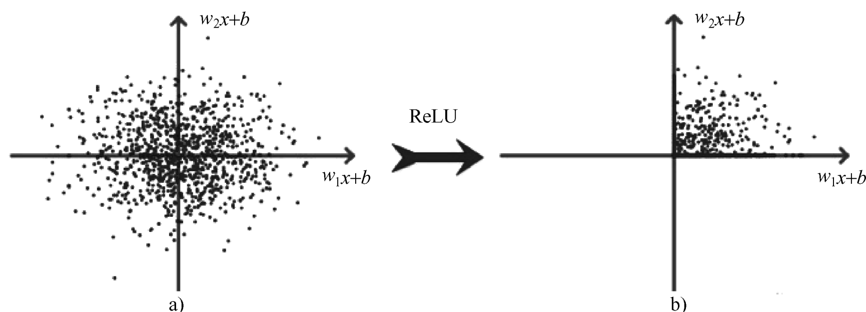


图 A.7　ReLU 效果

这主要是因为它是线性的，而且是非饱和的（因为 ReLU 的导数始终是 1），相比于 Sigmoid/tanh，ReLU 只需要一个阈值就可以得到激活值，而不用去算一大堆复杂的运算。

关于激活函数更多内容，请参考相关文献。

3） Dropout。

结合预先训练好的许多不同模型来进行预测是一种非常成功的减少测试误差的方式（Ensemble）。但因为每个模型的训练都需要花好几天时间，因此这种做法对于大型神经网络来说太过昂贵。

然而，AlexNet 提出了一个非常有效的模型组合版本，它在训练中只需要花费两倍于单模型的时间。这种技术叫作 Dropout，它做的就是以 0.5 的概率，将每个隐层神经元的输出设置为零。以这种方式“dropped out”的神经元既不参与前向传播，也不参与反向传播。

所以每次输入一个样本，就相当于该神经网络尝试了一个新的结构，但是所有这些结构之间共享权重。因为神经元不能依赖于其他特定神经元而存在，所以这种技术降低了神经元复杂的互适应关系。

正因如此，网络需要被迫学习更为鲁棒的特征，这些特征在结合其他神经元的一些不同随机子集时有用。在测试时，将所有神经元的输出都仅仅只乘以 0.5，对于获取指数级 Dropout 网络产生的预测分布的几何平均值，这是一个合理的近似方法。

4） 多 GPU 训练。

单个 GTX 580 GPU 只有 3 GB 内存，这限制了可以在其上训练的网络的最大规模。因此一般将网络分布在两个 GPU 上。

目前的 GPU 特别适合跨 GPU 并行化，因为它们能够直接从另一个 GPU 的内存

中读出和写入，不需要通过主机内存。

目前采用的并行方案是：在每个 GPU 中放置一半核（或神经元）。还有一个额外的技巧：GPU 间的通信只在某些层进行。

在图 A.2 中，第 3 层的核需要从第 2 层中所有核映射输入。然而，第 4 层的核只需要从第 3 层中位于同一 GPU 的那些核映射输入。

5）局部响应归一化（Local Responce Normalization，LRN）。

一句话概括：本质上，这个层也是为了防止激活函数的饱和的。基本原理是通过正则化让激活函数的输入靠近“碗”的中间（避免饱和），从而获得比较大的导数值。

所以从功能上说，与 ReLU 是重复的。从试验结果看，LRN 操作可以提高网络的泛化能力，将错误率降低大约 1 个百分点。

AlexNet 优势在于：网络增大（5 个卷积层 + 3 个全连接层 + 1 个 Softmax 层），同时解决了过拟合（Dropout，Data Augmentation，LRN），并且利用多 GPU 加速计算。

A.4 Overfeat

2013 年的 12 月，纽约大学的 Yann LeCun 实验室提出了 AlexNet 的衍生——Overfeat（参见：OverFeat：Integrated Recognition，Localization and Detection using Convolutional Networks）。这篇文章也提出了学习边界框（Learning Bounding Box），并导致之后出现了很多研究这同一主题的论文。目前学习分割对象比学习人工边界框更好。

A.5 VGG

来自牛津大学的 VGG 网络（参见：Very Deep Convolutional Networks for Large - Scale Image Recognition）是第一个在各个卷积层使用更小的 3 × 3 过滤器（Filter），并把它们组合作为一个卷积序列进行处理的网络。

这看来和 LeNet 的原理相反，其中是大的卷积被用来获取一张图像中相似特征。和 AlexNet 的 9 × 9 或 11 × 11 过滤器不同，过滤器开始变得更小，离 LeNet 所要避免的 1 × 1 卷积异常接近——至少在该网络的第一层是这样。但是 VGG 巨大的进展是通过依次采用多个 3 × 3 卷积，能够模仿出更大的感知效果，例如 5 × 5 与 7 × 7。这些思想也被用在了最近更多的网络架构中（如 Inception 与 ResNet）。网络配置见表 A.1 和表 A.2。

表 A.1　网络配置

<table>
<tr><td>A</td><td>A－LRN</td><td>B</td><td>C</td><td>D</td><td>E</td></tr>
<tr><td>11 层</td><td>11 层</td><td>13 层</td><td>16 层</td><td>16 层</td><td>19 层</td></tr>
<tr><td colspan="6">input(224×224) RGB image</td></tr>
<tr><td>conv3－64</td><td>conv3－64
LRN</td><td>conv3－64
conv3－64</td><td>conv3－64
conv3－64</td><td>conv3－64
conv3－64</td><td>conv3－64
conv3－64</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conc3－128</td><td>conc3－128</td><td>conc3－128
conc3－128</td><td>conc3－128
conc3－128</td><td>conc3－128
conc3－128</td><td>conc3－128
conc3－128</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3－256
conv3－256</td><td>conv3－256
conv3－256</td><td>conv3－256
conv3－256</td><td>conv3－256
conv3－256
conv1－256</td><td>conv3－256
conv3－256
conv3－256</td><td>conv3－256
conv3－256
conv3－256
conv3－256</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3－512
conv3－512</td><td>conv3－512
conv3－512</td><td>conv3－512
conv3－512</td><td>conv3－512
conv3－512
conv1－512</td><td>conv3－512
conv3－512
conv3－512</td><td>conv3－512
conv3－512
conv3－512
conv3－512</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td>conv3－512
conv3－512</td><td>conv3－512
conv3－512</td><td>conv3－512
conv3－512</td><td>conv3－512
conv3－512
conv1－512</td><td>conv3－512
conv3－512
conv3－512</td><td>conv3－512
conv3－512
conv3－512</td></tr>
<tr><td colspan="6">maxpool</td></tr>
<tr><td colspan="6">FC－4096</td></tr>
<tr><td colspan="6">FC－4096</td></tr>
<tr><td colspan="6">FC－1000</td></tr>
<tr><td colspan="6">soft－max</td></tr>
</table>

表 A.2　网络参数个数

网　　络	A,A－LRN	B	C	D	E
参数个数	133	133	134	138	144

VGG 网络使用多个 3×3 卷积层来表征复杂特征。注意 VGG－E 的第 3、4、5 块（Block）：256×256 和 512×512 个 3×3 过滤器被依次使用多次以提取更多复杂特征以及这些特征的组合。其效果等同于一个带有 3 个卷积层的大型的 512×512 大分类器。这显然意味着有大量的参数，使网络训练很困难，必须划分为较小的网络，并逐层累加。这是因为缺少对模型进行正则化的方法。

VGG 在许多层中都使用大特征尺寸，因为推断（Inference）在运行时是相当耗费时间的。正如 Inception 的瓶颈（Bottleneck）那样，减少特征的数量将节省一些计

算成本。

A.6 网络中的网络（Network－in－Network，NiN）

NiN 的思路简单又伟大：使用 1×1 卷积为卷积层的特征提供更组合性的能力。

NiN 架构在各个卷积之后使用空间 MLP 层，以便更好地在其他层之前组合特征，如图 A.8 所示。同样，可以认为 1×1 卷积与 LeNet 最初的原理相悖，但事实上它们可以以一种更好的方式组合卷积特征，而这是不可能通过简单堆叠更多的卷积特征做到的。这和使用原始像素作为下一层输入是有区别的。其中 1×1 卷积常常被用于在卷积之后的特征映射上对特征进行空间组合，所以它们实际上可以使用非常少的参数，并在这些特征的所有像素上共享。

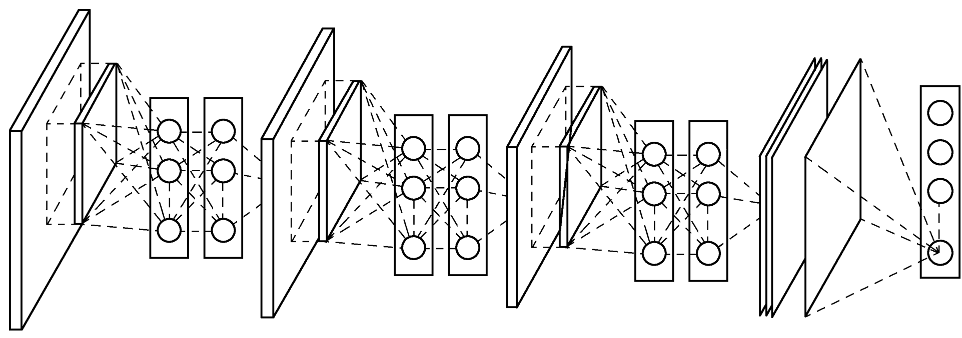


图 A.8　NiN 架构

MLP 的能力在于能通过将卷积特征组合到更复杂的组（Group）来极大地增加单个卷积特征的有效性。这个想法之后被用到一些最近的架构中，例如 ResNet、Inception 及其衍生技术。

NiN 也使用了平均池化层作为最后分类器的一部分，这是另一种将会变得常见的实践。这是通过在分类之前对网络对多个输入图像的响应进行平均完成的。

A.7 GoogLeNet 与 Inception

来自谷歌的 Christian Szegedy 开始追求减少深度神经网络的计算开销，并设计出 GoogLeNet——第一个 Inception 架构（参见：Going Deeper with Convolutions）。

2014 年秋季，深度学习模型在图像与视频帧的分类中变得非常有用。鉴于这些技术的用处，谷歌这样的互联网巨头非常有兴趣在他们的服务器上高效且大规模庞大地部署这些架构。

Christian 考虑了很多关于在深度神经网络达到最高水平的性能（例如在 ImageNet 上）的同时减少其计算开销的方式；或者在能够保证同样的计算开销的前提下对性能有所改进。

他和他的团队提出了 Inception 模块（见图 A.9）。

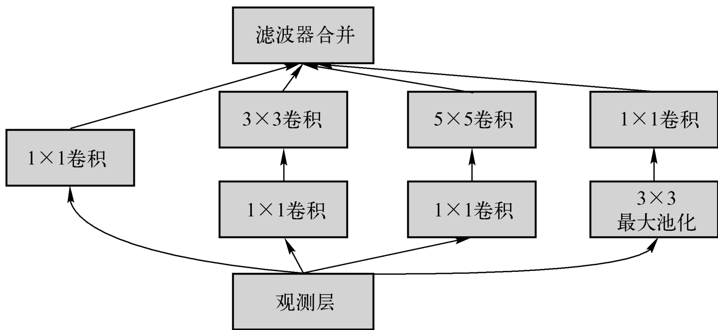


图 A.9　Inception 模块

初看之下这不过是 1×1、3×3、5×5 卷积过滤器的并行组合。但是 Inception 的伟大思路是用 1×1 的卷积块（NiN）在昂贵的并行模块之前减少特征的数量。这一般被称为瓶颈（Bottleneck）。这部分内容将在下面的瓶颈层（Bottleneck Layer）部分来解释。

GoogLeNet 使用没有 Inception 模块的主干作为初始层，之后是与 NiN 相似的一个平均池化层加 Softmax 分类器。这个分类器比 AlexNet 与 VGG 的分类器的运算数量少得多。

受到 NiN 的启发，Inception 的瓶颈层（Bottleneck Layer）减少了每一层的特征的数量，并由此减少了运算的数量；所以可以保持较低的推理时间。在将数据通入昂贵的卷积模块之前，特征的数量会减少 4 倍。在计算成本上这是很大的节约，也是该架构的成功之处。

下面具体验证一下：现在有 256 个特征输入，256 个特征输出，假定 Inception 层只能执行 3×3 的卷积，也就是总共要完成 256×256×3×3 的卷积（将近 589 000 次乘积累加（MAC）运算）。这可能超出了计算预算，比如，在谷歌服务器上要以 0.5 ms 运行该层。作为替代，减少需要进行卷积运算的特征的数量，也就是 64（即 256/4）个。在这种情况下，首先进行 256 ->64 1×1 的卷积，然后在所有 Inception 的分支上进行 64 次卷积，接而再使用一个来自 64 ->256 的特征的 1×1 卷积，运算如下：

$$256 \times 64 \times 1 \times 1 = 16\,000\ \mathrm{s}$$

$$64 \times 64 \times 3 \times 3 = 36\,000\ \mathrm{s}$$

$$64 \times 256 \times 1 \times 1 = 16\,000\ \mathrm{s}$$

相比于之前的 60 万，现在共有 7 万的计算量，几乎减少了近 10 倍。

而且，在获得了更好的运算的同时此层并没有损失其通用性（Generality）。事实证明，瓶颈层在 ImageNet 这样的数据集上已经表现出了顶尖水平，而且它也被用于接下来介绍的 ResNet 这样的架构中。

它之所以成功是因为输入特征是相关联的，因此可通过将它们与 1×1 卷积适当结合来减少冗余。然后，在小数量的特征进行卷积之后，它们能在下一层被再次扩展成有意义的结合。

A.8　Inception V3（还有 V2）

Christian 和他的团队都是非常高产的研究人员。2015 年 2 月，Batch-normalized Inception 被引入作为 Inception V2（参见论文：Batch Normalization：Accelerating Deep Network Training by Reducing Internal Covariate Shift）。Batch-normalization 在一层的输出上计算所有特征映射的均值和标准差，并且使用这些值规范化它们的响应。这相当于数据增白（Whitening），因此使得所有神经图（Neural Maps）在同样范围有响应，而且是零均值。在下一层不需要从输入数据中学习 Offset 时，这有助于训练，还能重点关注如何最好地结合这些特征。

2015 年 12 月，该团队发布 Inception 模块和类似架构的一个新版本（参见论文：Rethinking the Inception Architecture for Computer Vision）。该论文更好地解释了原始的 GoogLeNet 架构，在设计选择上给出了更多的细节。原始思路如下：

1）通过谨慎建筑网络，平衡深度与宽度，从而最大化进入网络的信息流。在每次池化之前，增加特征映射。

2）当深度增加时，网络层的深度或者特征的数量也系统性地增加。

3）当深度增加时同时增加特征。

4）只使用 3×3 的卷积，可能的情况下给定的 5×5 和 7×7 过滤器能分成多个 3×3（见图 A.10）。

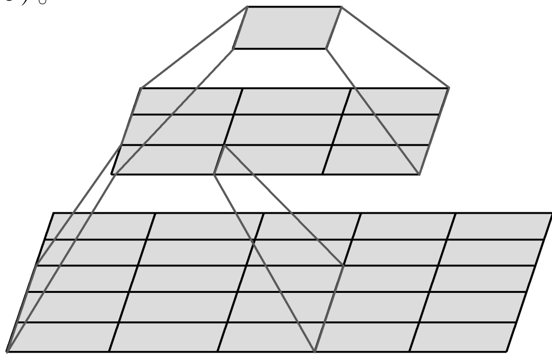


图 A.10　Inception V3 架构

因此新的 Inception 变成图 A. 11 所示。

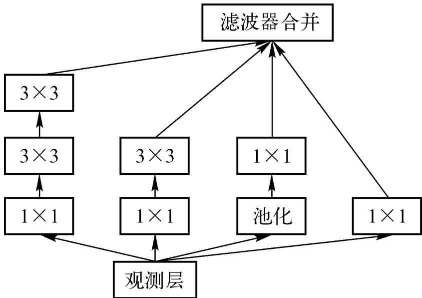


图 A. 11　Inception V3 过滤技术

也可以通过将卷积平整到更多复杂的模块中而分拆过滤器，如图 A. 12 所示。

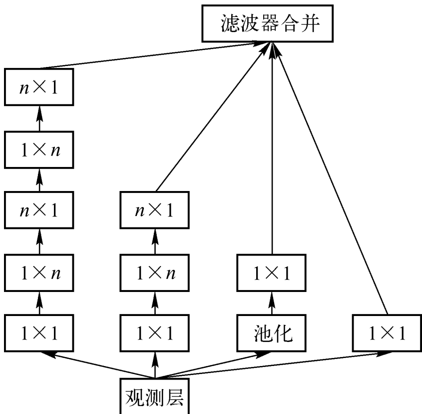


图 A. 12　Inception V3 过滤器拆分

在进行 Inception 计算的同时，Inception 模块也能通过提供池化降低数据的大小。这基本类似于在运行一个卷积的时候并行一个简单的池化层，如图 A. 13 所示。

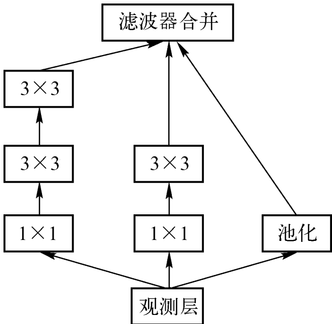


图 A. 13　Inception V3 池化技术

Inception 也使用一个池化层和 Softmax 作为最后的分类器。

A.9　ResNet

2015 年 12 月又出现了新的变革，这和 Inception V3 出现的时间一样。ResNet 有着简单的思路：供给两个连续卷积层的输出，并分流（Bypassing）输入进入下一层，如图 A.14 所示，（参见论文：Deep Residual Learning for Image Recognition）。

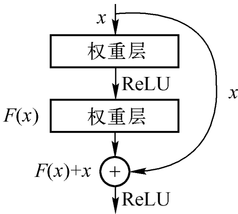


图 A.14　ResNet 结构

这和之前的一些旧思路类似。但 ResNet 中，它们分流两个层并被应用于更大的规模。在 2 层后分流是一个关键，因为分流一个层并未给出更多的改进。通过 2 层可能认为是一个小型分类器，或者一个 Network – in – Network。

这是第一次网络层数超过 100，甚至还能训练出 1000 层的网络。

有大量网络层的 ResNet 开始使用类似于 Inception 瓶颈层的网络层，如图 A.15 所示。

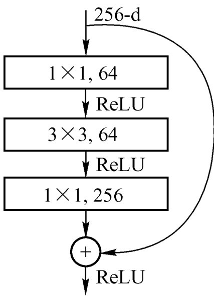


图 A.15　ResNet 的瓶颈层的网络层

这种层通过首先使用带有更小输出（通常是输入的 1/4）的 1 ×1 卷积较少特征的数量，然后使用一个 3 ×3 的层，再使用 1 ×1 的层处理更大量的特征。类似于 Inception 模块，这样做能保证计算量低，同时提供丰富的特征结合。

ResNet 在输入上使用相对简单的初始层：一个带有两个池的 7 ×7 卷积层。可以把这个与更复杂、更少直觉性的 Inception V3、V4 做下对比。

ResNet 也使用一个池化层加上 Softmax 作为最后的分类器。

关于 ResNet 的其他见解如下：

ResNet 可被认为既是并行模块又是连续模块，把输入输出（Inout）视为在许多模块中并行，同时每个模块的输出又是连续连接的。

ResNet 也可被视为并行模块或连续模块的多种组合（参见论文：Residual Networks are Exponential Ensembles of Relatively Shallow Networks）。

已经发现，ResNet 通常在 20～30 层的网络块上以并行的方式运行，而不是连续流过整个网络长度。

当 ResNet 像 RNN 一样把输出反馈给输入时，该网络可被视为更好的生物上可信的皮质模型（参见论文：Bridging the Gaps Between Residual Learning，Recurrent Neural Networks and Visual Cortex）。

A. 10 Inception V4

这是 Christian 与其团队的另一个 Inception 版本，该模块类似于 Inception V3，如图 A. 16 所示。

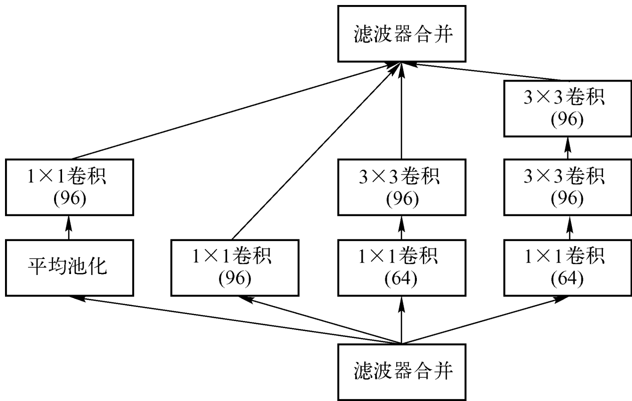


图 A. 16 Inception V4 模块 1

Inception V4 也结合了 Inception 模块和 ResNet 模块，如图 A. 17 所示。

A. 11 SqueezeNet

SqueezeNet（参见论文：SqueezeNet：AlexNet – Level Accuracy with 50x Fewer Parameters and <0. 5 MB Model Size）是最近才公布的，该架构是对 ResNet 与 Inception 里面概念的重新处理。一个更好的架构设计网络型号要小，而且参数还不需要复杂的压缩算法。

A. 12 ENet

ENet 网络架构由 Adam Paszke 设计，目前已经使用它进行过单像素标记和场景解析。

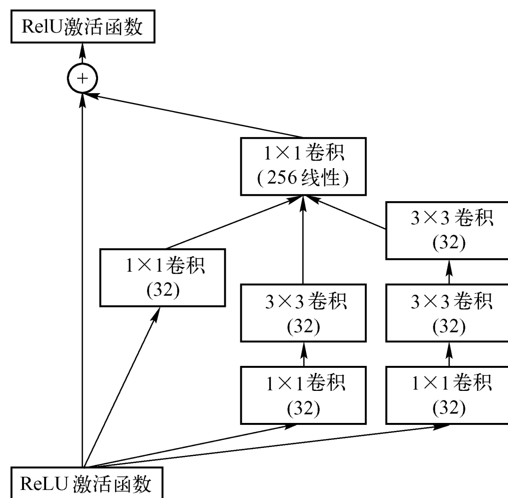


图 A. 17 Inception V4 模块 2

详细了解 ENet 可参见论文 ENet：A Deep Neural Network Architecture for Real – Time Semantic Segmentation。ENet 是一个编码加解码的网络。编码器是一个常规的 CNN 设计进行分类。解码器是一个增采样（Upsampling）网络，将分类反向传播给原始图像进行分割。这只使用了神经网络，没有其他算法进行图像分割。

ENet 被设计为在开始时尽可能使用最小数量的资源。正是如此它有着如此小的脚本，编码器和解码器网络共占有 0. 7 MB，16 fp 精度。即使这么小的型号，ENet 在分割的准确度上也类似于或者高于其他神经网络解决方案。

A. 13 其他值得关注的架构

FractalNet（参见论文：FractalNet：Ultra – Deep Neural Networks without Residuals）使用递归架构，它在 ImageNet 上没有进行测试。该架构是 ResNet 的衍生或者更通用的 ResNet。

制作神经网络架构是深度学习领域发展的头等大事。推荐仔细阅读并理解本节中提到的论文。

但有人可能会想为什么要投入如此多的时间制作架构？为什么不是用数据告诉我们使用什么？如何结合模块？这些问题很好，但仍在研究中，可以参考论文：Neural Networks with Differentiable Structure。

要注意到，在本书中提到的大部分架构都是关于计算机视觉的。类似神经网络架构在其他领域内也有开发，学习其他所有任务中的架构变革也是非常有趣的。

如果对神经网络架构和计算性能的比较有兴趣，可参见论文：An Analysis of Deep Neural Network Models for Practical Applications。

附录 B　深度学习的未来——GAN

B.1　对抗样本

2014 年 Szegedy 在研究神经网络的性质时，发现针对一个已经训练好的分类模型，将训练集中样本做一些细微的改变会导致模型给出一个错误的分类结果，这种虽然发生扰动但是人眼可能识别不出来，并且会导致误分类的样本被称为对抗样本，他们利用这样的样本发明了对抗训练（Adversarial Training），模型既训练正常的样本也训练这种自己造的对抗样本，从而改进模型的泛化能力。如图 B.1 所示，在未加扰动之前，模型认为输入图片有 57.7% 的概率为熊猫，但是加了之后，人眼看着好像没有发生改变，但是模型却认为有 99.3% 的可能是长臂猿。

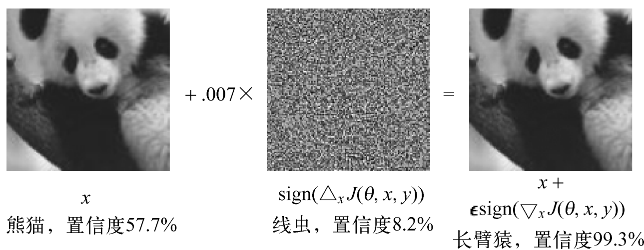


图 B.1　对抗样本的产生

Goodfellow 在 2015 年指出，其实模型欠拟合也能导致对抗样本，因为从现象上来说是输入发生了一定程度的改变从而导致了输出的不正确，如图 B.2 a、b 分别是过拟合和欠拟合导致的对抗样本，其中灰色的 o 和 x 代表训练集，黑色的 o 和 x 即对抗样本，可以明显看到，欠拟合的情况下输入发生改变也会导致分类不正确（这里图 B.2 中所描述的对抗样本不一定与原始样本是同分布的，也可能是人为造的一个东西，而不是真实数据的反馈）。

B.2　生成式对抗网络 GAN

生成式对抗网络（Generative Adversarial Network，GAN）启发自博弈论中的二人零和博弈，将成为深度学习的下一个热点，它将改变我们认知世界的方式。

准确来讲，对抗式训练为指导人工智能完成复杂任务提供了一个全新的思路，某种意义上他们（人工智能）将学习如何成为一个专家。

GAN 解决问题的方式是用不同的目标分别训练两种不同的网络：

1）创造答案（Generator，生成方或生成网络）。

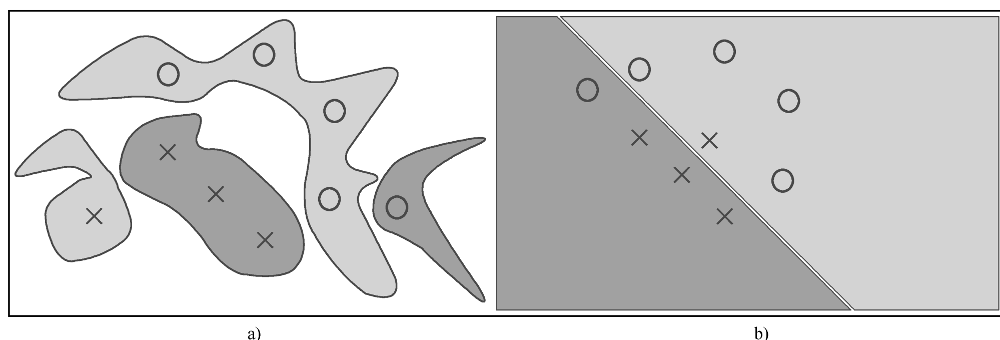


图 B.2　过/欠拟合导致对抗样本

2）分辨创造答案与真实答案的区别（Discriminator，对抗方或判断网络）。

GAN 的思想是：训练两种网络进行竞争，一段时间后，两种网络都无法在对抗中取得进步，或者生成方变得非常厉害以至于即使给定足够的线索和时间，其对抗网络也无法分辨它给的答案是真实的还是合成的。

GAN 要解决的问题是如何从训练样本中学习出新样本，训练样本是图片就生成新图片，训练样本是文章就输出新文章等。

举个对抗式训练的例子，当你试图通过模仿别人完成某项工作时，如果专家都无法分辨这项工作是你完成的还是你的模仿对象完成的，说明你已经完全掌握了该工作所需的技巧。对于像写论文这样复杂的工作，这个例子可能不适用，毕竟每个人的最终成果多少有些不同，但对于造句或写一段话，对抗式训练大有用武之地，事实上它现在已经是计算机生成真实图像的关键所在了。算法流程如图 B.3 所示。

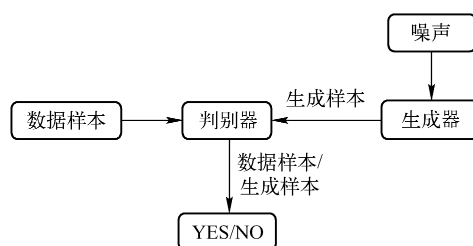


图 B.3　GAN 算法

B.3　对抗学习

相比强化学习（Reinforcement Learning，RL）而言，对抗式学习更接近人类的学习方式。

RL 通过最大化（平均）最终奖励来达到训练目的。当前的状态也许与奖励无关，但最终的结局一定会由“奖励函数”给出。

20 世纪 90 年代，强化学习在十五子棋游戏中取得巨大突破，它是 DeepMind 创造的 AlphaGo 的一个重要组成部分，DeepMind 团队甚至用 RL 来节省谷歌的数据中心的冷却费用。

RL 在谷歌数据中心环境中，可以很好地定义奖励函数（在防止温度高于限定值的条件下尽可能省钱）。

对于那些更实际的问题，奖励函数是什么呢？即使是类似游戏中的任务如驾驶，其目标既不是尽快到达目的地，也并非始终待在道路边界线内。可以很容易地找到一个负奖励（比如撞坏车辆、使乘客受伤、不合理地加速），但却很难找到一个可以规范驾驶行为的正奖励。

B.4 应用实例

我们是如何学习写字的？除非是在要求很严格的小学，否则学习写字的过程很难说是最大化某个与书写字母有关的函数。最可能的情况是你模仿老师在黑板上的书写笔顺，然后内化这一过程。

生成网络书写字母，而识别网络（对抗方）观察书写的字体和教科书中理想字体的区别，如图 B.4 所示。



图 B.4 针对三年级学生的对抗式训练

现实中通常是老师更会写字，但通过这个例子，应该是学生更会写字。因为一个用于识别产品评论的识别器只需知道一些常见的语法错误，就能投入使用。

所以，可以得出一个很自然的结论，生成式对抗方法，可以让人工智能有能力

做实验和 A/B 测试。一个人工智能模型给出了一个很好的解决方案，然后它搜集反馈来了解这个生成方案和标准答案是否相符，或与其他正在学习或已经内化的人工智能模型比较，观察得出的结果是否相符。在这一过程中，没必要去设计一个损失函数，因为虽然可能会花上一点时间，但这个人工智能模型终将找到自己的评判标准。

B.5 GAN 优势

GAN 优势如下：

1）非常好的特征学习。

2）在强化学习中的探索。

3）逆向增强学习。

4）真正使用的对话。

5）理解世界。

6）迁移学习。

附录 C　R 包分类

（1）聚类

常用的包：fpc，cluster，pvclust，mclust

基于划分的方法：kmeans，pam，pamk，clara

基于层次的方法：hclust，pvclust，agnes，diana

基于模型的方法：mclust

基于密度的方法：dbscan

基于画图的方法：plotcluster，plot. hclust

基于验证的方法：cluster. stats

（2）分类

常用的包：rpart，party，randomForest，rpartOrdinal，tree，marginTree，maptree，survival

决策树：rpart，ctree

随机森林：cforest，randomForest

回归：Logistic，Poisson，glm，predict，residuals

生存分析：survfit，survdiff，coxph

（3）序列模式

常用的包：arulesSequences，timsac，zoo

SPADE 算法：cSPADE

常用的包：timsac

时间序列构建函数：ts

成分分解：decomp，decompose，stl，tsr

（4）统计

常用的包：Base R，nlme

方差分析：aov，anova

密度分析：density

假设检验：t. test，prop. test，anova，aov

线性混合模型：lme

主成分分析和因子分析：princomp

方差分析对应的是 Kruskal – Wallis 秩和检验：kruskal. test

T 检验对应的是 Wilcoxon 符号秩和检验 R：wilcox. test

随机微分方程模拟和统计推断：sde

非参数平滑与分布估计：KernSmooth

用来方便地做 MLE 估计：stats4

非线性方程求根、ODE 均衡状态解：rootSolve

非线性优化：Rsolnp

（5）机器学习

神经网络：nnet，deepnet，RcppDL，autoencoder，neuralnet，RSSNS，Rdbn，AMORE，MXNetR，darch，H2O

支持向量机 SVM：e1071

核函数：kernlab

（6）数据库

连接 ODBC 数据库接口：RODBC

连接轻量级 SQLite 数据库连接：RSQLite

像写 SQL 语句一样对数据框做操作：sqldf

（7）NLP

词汇数据库：Wordnet

命名实体识别：openNLP，KoNLP

分词：Snowball/Rwordseg（分词 http://R－Forge. R－project. org）

词干化：Rstem

文本挖掘：RTextTool，textact，Isa，tm

主题模型：topicmodel

关键词提取：tau，gsubfn，RKEA

参考文献

深度学习经典论文：

[1] Hinton G E, Salakhutdinov R. Reducing the dimensionality of data with neural networks[J]. Science, 2006:504 – 507. 313(5786).

[2] Bengio Y. Learning deep architectures for AI[J]. Machine Learning, 2009, 2(1):1 – 127.

[3] Arel I, Rose D C, Karnowski T P. Deep machine learning—A new frontier in artificial intelligence research[J]. IEEE Computational Intelligence Magazine, 2010:13 – 19.

[4] Bengio Y, Courville A, Vincent P. Representation learning: A review and new perspectives[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2012:1 – 34.

[5] 深度学习大讲堂[J/OL]. http://blog. csdn. net/lg1259 156776/article/details/ 52551158.

R 语言学习：

[6] R 语言官方网站[J/OL][2015 – 02 – 12]. http://cran. csdb. cn/index. html.

[7] 统计之都论坛[J/OL][2014 – 07 – 05]. http://cos. name/.

[8] 国外著名的 R 语言群博[J/OL][2016 – 04 – 15]. http://www. r – bloggers. com/.

[9] R 语言可视化[J/OL][2015 – 02 – 18]. http://addictedtor. free. fr/graphiques/.

[10] R 包干货[J/OL]. http://blog. sina. com. cn/s/blog_15dd952a70102wvuq. html.

数据科学：

[11] 2016 数据科学报告：数据科学家依然受追捧[J/OL][2016 – 09 – 28]. http://blog. csdn. net/u013886628/article/details/51820196.

[12] 课课家[J/OL][2015 – 10 – 13]. http://www. kokojia. com/course – 3831. html.

[13] 雪睛数据网[J/OL][2014 – 11 – 14]. http://www. xueqing. tv/course.

深度学习应用：

[14] 深度学习应用大盘点[J/OL][2016 – 03 – 15]. http://blog. csdn. net/zhaoyu106/ article/details/53261055.

[15] 深度学习在图像识别成功应用的例子

- Krizhevsky A, Sutskever I, Hinton G E. Imagenet classication with deep convolutional neural networks [J]. Advances in Neural Information Processing Systems, 2012: 1097 – 1105.

● Keshmiri S, Xin Z, Lu W F. Application of deep neural network in estimation of the weld bead parameters [C]. IEEE/RSJ International Conference on Intelligent Robots & Systems, Hamburg, 2015: 3518 – 3523.

● Dahl G E, Dong Y, Li D, et al. Context – dependent pre – trained deep neural networks for large – vocabulary speech recognition [J]. IEEE Transactions on Audio Speech & Language Processing, 2012, 20 (1): 30 – 42.

[16] 其他领域深度学习应用的例子

● Payan A, Montana G. Predicting Alzheimer's disease: A neuroimaging study with 3D convolutional neural networks [J]. Computer Science, 2015.

● Oko E, Wang M H, Zhang J. Neural network approach for predicting drum pressure and level in coal – fired subcritical power plant [J]. Fuel, 2015: 139 – 145.

● Torre A, Garcia F, Moromi I, et al. Prediction of compression strength of high performance concrete using artificial neural networks [J]. Journal of Physics: Conference Series, 2015.

R 语言和深度学习:

[17] Lewis N D. Deep Learning Made Easy with R—A Gentle Introduction for Data Science [M]. Charleston: Greate Space Independent Publishing, 2016.

[18] R 语言和深度学习[J/OL]. http://blog. csdn. NET/easonlv/article/details/23427809.

[19] darch[J/OL][2015 – 06 – 16]. http://cran. um. ac. ir/web/packages/darch/index. html.

[20] deepnet [J/OL] [2015 – 11 – 06]. http://cran. r – project. org/web/packages/deep – net/index. html.

[21] Rdbn[J/OL][2015 – 12 – 04]. https://github. com/dankoc/Rdbn.

[22] R 语言结合 H2O 做深度学习[J/OL][2015 – 12 – 16]. http://blog. itpub. Net/16582684/viewspace – 1255976/.

[23] MXNetR, 原生态 R 语言深度学习, 支持 GPU 计算[J/OL][2016 – 04 – 09]. https://github. com/dmlc/mxnet/tree/master/R – package.

神经网络:

[24] 神经网络演进史全面回顾[J/OL][2014 – 04 – 12]. http://www. open – open. com/lib/view/open1473213789568. html.

CNN:

[25] 胡邵华, 宋耀良. 基于 autoencoder 网络的数据降维与重构[J]. 电子与信息学报, 2009, 31 (5): 1189 – 1192.

MXnet:

[26] mxnet 官网[J/OL][2016 – 02 – 13]. http://mxnet. io/get_started/setup. html#install – ing – mxnet – on – a – gpu

[27] 结合 Shiny + MXNet 搭建在线识图服务[J/OL][2016 - 02 - 08]. http://dmlc. ml/rstats/2015/12/08/image - classification - shiny - app - mxnet - r. html.

[28] mxnet R 包入门文档[J/OL][2016 - 07 - 11]. http://mxnet. io/tutorials/index. html#rtutorials.

[29] MXNet 完整文档[J/OL][2016 - 09 - 23]. http://mxnet. readthedocs. io/en/latest/.

[30] MXNet on github[J/OL][2016 - 09 - 26]. https://github. com/dmlc/mxnet.

Word2vec:

[31] Windows 下使用 Word2vec 进行词向量训练[J/OL][2016 - 02 - 08]. http://blog. csdn. net/heyongluoyao8/article/details/43488765.

[32] word2vec——高效 word 特征求取[J/OL][2015 - 08 - 18]. http://blog. csdn. net/abcjennifer/article/details/46397829.

[33] 周练. Word2vec 的工作原理及应用探究[D]. 西安：西安电子科技大学，2014.

[34] Yakovenko N. 深度学习的下一个热点——GANs 将改变世界[J/OL][2017 - 01 - 10]. http://chenrudan. github. io.

后　记

实践深度学习应注意以下问题：

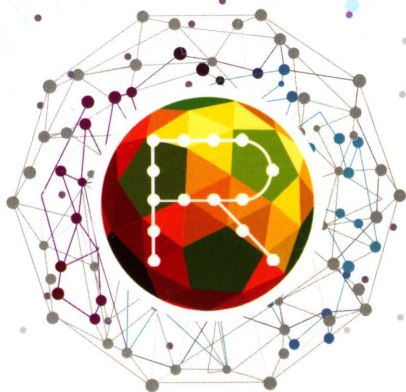
第一点，在学习深度学习之前，总以为深度学习是很了不得的知识，能解决很多问题。其实不然，深度学习算法与其他机器学习算法（如 SVM 等）相似，仍然可以把深度学习算法当作一个分类器，像使用一个黑盒子那样使用深度学习算法。

第二点，深度学习强大的地方就是可以利用网络中间某一层的输出当作数据的另一种表达，从而可以认为深度学习是经过网络学习到的特征。基于该特征，可以进行进一步的相似度比较等。

第三点，深度学习算法能够有效的关键是使用大规模的数据，原因在于每个深度学习算法都有众多的参数，少量数据无法将参数训练充分。

在线互动交流平台

官方微博：http://weibo.com/cmpjsj
豆 瓣 网：http://site.douban.com/139085/
读者信箱：cmp_itbook@163.com



深度学习与R语言

基于R语言的深度学习不需要很深的数学作为前提；
通过案例让读者一步一步构建和部署深度学习模型；
可以用最少的时间开发深度学习应用系统


IT有得聊

关注计算机分社官方微信，回复您购买图书书号中间的五位数字，即可获取本书配套资源下载链接，并可获得更多增值服务和最新资讯。

ISBN 978-7-111-**57073**-8

地址：北京市百万庄大街22号
邮政编码：100037
电话服务
服务咨询热线：010-88361066
读者购书热线：010-68326294
010-88379203
网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
金 书 网：www.golden-book.com
教育服务网：www.cmpedu.com
封面无防伪标均为盗版


机械工业出版社
微信公众号

上架指导 人工智能/深度学习

ISBN 978-7-111-57073-8

策划编辑◎ 汤 枫 / 封面设计◎  子时文化 ZiShi Culture



定价：49.00元